A PROGRAMMABLE CELL FOR A SYSTOLIC ARRAY

A Thesis Submitted
in Partial Fulfilment of the Requirements
for the Degree of

MASTER OF TECHNOLOGY

by

S. R. SUBRAMANIAN

to the

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR
MARCH, 1990

22 3 90

CERTIFICATE

This is to certify that the thesis work entitled " A Programmable Cell for a Systolic Array " has been carried out by Mr S R Subramanian under our supervision and the same has not been submitted elsewhere for a degree

R N Biswas
Professor
Dept of EE
IIT, Kanpur

Anil Mahanta
Asst Professor
Dept of EE
IIT, Kanpur

March 1990

24 JAN 1991

CENTRAL LIBRARY

No. 109950

EE-1990-M-SUB-PRO

ACKNOWLEDGEMENTS

I express my sincere gratitude to my guides Dr Anil Mahanta and Dr R N Biswas, for their guidance throughout this work

I am grateful to Dr Anil Mahanta for introducing me to the concepts of parallel processing and digital signal processing and for initiating me to take up this thesis work I am grateful to Dr R N Biswas for his expert guidance and suggestions during the design and implementation of the hardware I would also like to thank him for providing me with a project fellowship during the last few months of my M Tech programme

I would like to acknowledge the permission of Dr K R Srivathsan and Dr Sanjay K Bose for letting me use the MDS Lab facilities I am also grateful to Nitin for all the help he has extended to me during all the plotting sessions Also, thanks are due to Mr S S Bhatnagar, Seshadiri, Ghatare and other Research Engineers for their cooperation and useful suggestions

I am also grateful to my friends Ganesan, Sanjay, Shera, Ravindra and Kasi for their company and the help I received from them

Finally I take pride in acknowledging the cheerful company of my hostel friends Kumar, Seetharam, Raghavan and Mansoor who made my stay at I I T, Kanpur a memorable one

CONTENTS

1. INTRODUCTION

1.1 SIGNAL PROCESSING PAST AND PRESENT

Classical signal processing techniques utilized analog technology and was characterized by static realizations of low-pass, band-pass, and high-pass filters based on the knowledge of signal and noise spectra. The end result of such analog computation is prone to the error due to noise, drift with temperature and aging of resistors, capacitors, etc., used in the design of these filters. Limited dynamic range and finite signal-to-noise ratio restricted signal processing techniques with analog technology. Further for many applications this approach was not feasible because it was too difficult to optimize the performance of the system with too many design parameters. This then lead to the use of digital computers in signal processing applications. In the earlier stages digital computer was not used for real-time signal processing applications. The primary operations involved were filtering, convolution, correlation, FFT etc. The computational complexity were $O(N\log_2(N))$ for FFT and $O(N^2)$ for other operations. Theoretical advances in signal processing brought in newer concepts like adaptive filters, Kalman filters, linear predictive coding model for speech. These concepts involved operations like matrix-vector multiplication, matrix-matrix multiplication, LU decomposition, linear system solution, singular value decomposition and Hermitian eigensystem solutions. The computational complexity of these operations are $O(N^3)$, thus putting great demand on effective computational means for realization of these systems. Fortunately the advances in signal processing theory was concomitant with the rapid advances in integrated circuit technology.

In the early days of digital computers, hardware was expensive and digital computers had limited capabilities With advances in hardware technology parallelism was introduced in the uniprocessor computer which was in the form of multiple functional units Different aspects of the parallelism are pipelining the CPU, overlapping CPU and I/O operation, hierarchical memory system, multiprogramming and time-sharing [14] As scientific computation occupied only a small fraction of the computer time, general purpose computers were not fine-tuned for scientific applications As science advanced so were the needs of the scientific community for fast and greater computing power The requirement of greater computational throughput can be achieved by (i) improvement in circuit performance, (ii) parallel processing

Improvements in circuit performance is largely driven by technology available at any given time Devices made of silicon are approaching the theoretical and technological limits for temperature and speed Even machines that use Josephson's devices or have logic devices made of gallium arsenide improve the performance over that of silicon only one to two order of magnitude Indeed, were the performance of the devices improved even beyond present limits, substantial improvement in the overall speed of the computer is not guaranteed

The ever increasing demands for performance and real-time signal processing strongly indicate the need for tremendous computational capability, both in terms of volume and speed This fact is also illustrated by the wide acceptance of vector processing machine like the Cray-1 The speedup in these machines has been achieved through architectural improvements, mentioned before, and developments of compilers to map programs designed for serial machines onto a vector architecture Despite the impressive speed of these computers, the von Neumann approach to their architecture limits their usefulness for computation intensive problems Achieving high performance depends not only on using faster

devices but also calls for a major improvement in computer architecture Throughput can be increased by parallel processing, in which multiple computing elements work concurrently on the solution of a single problem Concurrency implies pipelining, simultaneity and parallelism Such concurrency is available at various information processing level such as job or program level, task or procedure level, inter-instruction level and intra-instruction level Current parallel processing systems can be divided into three architectural configuration [14]

* Pipeline computer
* Multiprocessor systems
* Array processor

A pipeline computer exploits temporal parallelism by overlapping the execution of different instructions Multiprocessor systems achieves asynchronous parallelism through a set of interactive processors Finally, an array processor exploits spatial parallelism using multiple synchronized arithmetic logic units

1.2 PIPELINE COMPUTER

A pipeline computer is a SIMD processor which works according to the principle of pipelining The pipelining concept implies the partitioning of instructions into simpler computational steps which can be executed independently by computational units Due to the overlap of different instructions, pipeline machines are better tuned to perform the same operation repeatedly on a large set of data Therefore pipeline computers are more suited for vector processing and can be found in most computers designed for such applications Examples of such machines are the CDC STAR 100, Texas Instruments ASC, all Cray machines, Cyber-205, Fujitsu VP-200 and the attached pipeline processors AP-120B and FPS-

164 by Floating Point Systems and the IBM 3838

The limitations of pipelining techniques are -

i) When the vectors are short, the speedup is small because of the relatively large fraction of execution time is wasted in filling and draining the pipeline, which causes a delay before the initial results emerge from the pipeline

ii) Theoretically a k-stage linear pipeline processor can be at most k times faster Therefore improvement in speed based on pure pipelining alone is small

iii) It is difficult to implement either conditional branching or subroutine calls dependent on the data flowing through the pipe or operations among data while they are still in the pipe

1 3 MULTIPROCESSOR SYSTEMS

A basic multiprocessor system contains two or more processors having local memories and private devices and sharing access to common memory modules, I/O channels and peripherals devices The entire system is controlled by a single integrated operating system providing interactions between processors and their programs at various levels Interprocessor communication can be done by using - i) time-shared common bus, ii) crossbar switch network and iii) multiport memories and a variety of interconnection networks

1 3 1 Time Shared Common Bus

The time-shared common bus can be a single bus as in Intel's Multibus II, Motorola's VME bus, Texas Instrument's Nu Bus, the proposed IEEE 896 Futurebus and Digital Equipment's Unibus and Qbus [12] In a single time-shared bus, the performance is decided by the time spent by a processor waiting for the bus, the time duration for which the bus was available, and the bus capacity (in bytes per

second) Contentions for bus is bound to increase as the number of processor modules increases and thereby degrading the system performance Multiprocessor system is suited for coarse-grain parallelism problems which require a minimal amount of interprocessor communication One way to improve the system throughput is to connect the processors in a hierarchical cluster by two-level buses such that processors in a cluster communicate over one bus and the intercluster communication takes place on the other bus Systems based on hierarchical bus concepts include the Cm* of Carnegie-Mellon University and the Encore's Ultramax system

1 3.2 Crossbar Switch Network

The crossbar switch evolved through the attempt to overcome the potential throughput limitations of systems organized on a time-shared bus An example system of this type is the Carnegie Mellon University's C mmp system in which the memories and I/O nodes are connected to the processor modules A crossbar provides the highest bandwidth and the best performance for a interconnected system but at the expense of complexity, size and cost, which are proportional to the square of the number of interconnected components Expansion of the system is limited only by the size of the switch matrix that is feasible

1 3 3 Multiport Memories

If the control, switching and priority arbitration logic that is distributed throughout the crossbar switch matrix is redistributed at the interfaces to the memory modules, a multiport memory system results This system organization is well-suited to both uniprocessor and multiprocessor system organizations A common method to resolve memory-access conflicts is to assign permanently designated priorities at each memory port

It is very difficult to justify economically the use of a crossbar switch or multiport memories for large multiprocessing systems The cost of the switch can be reduced by using a switch with a restricted number of possible permutations These networks are less expensive than a full crossbar and multiport memories for large multiprocessing systems Moreover they are modular, and easy to control and expand

1 4 ARRAY PROCESSOR

The pipeline computer performs well for operations on long vectors Bus-based multiprocessor systems are better-suited for algorithms with coarse-grain parallelism The array processor is not only architecturally different, but performs exceptionally for certain kind of problems They are a cost-effective tool for increasing the speed of highly compute-bound processing

The term "array processor" has been attached to a variety of architectures Array processors include FPS's AP-120B, which has a single computational unit and the ILLIAC IV with its 64 computational units In this thesis the term *array processor* connotes an attached processor (connected to a host computer system) with multiple ALUs, called Processing Elements (PEs) that can operate in parallel in a lock-step fashion so that the tandem combination of the host and the array processor provides a much higher number-crunching capability than the host alone

To see the motivation for using an array processor, we need only consider the addition of two N X N matrices Here the N^2 additions can be performed completely in parallel in a single addition time, were there available N^2 adders Another example would be in image processing where identical transformations have to be carried out on every pixel, and this can be done for as many elements as there are processors

The idea of using regular array of processors appears to go back to Unger in 1958 The first machine actually designed with processor array architecture was the Solomon computer, along the lines proposed by Slotnick A much more powerful machine, based on Solomon's idea, the ILLIAC IV was produced ten years later The experience gained from ILLIAC IV led to the design of the Burrough's BSP which also had the feature of instruction streaming Other machines based on the original Unger's idea are the ICL's Distributed Array Processor and NASA's Massively Parallel Processor

1 5 VLSI ARRAY PROCESSORS

A very promising solution to the real-time requirement of signal processing is to use special-purpose array processors and to maximize the processing concurrency either by pipeline processing or parallel processing or both Such highly parallel computing structures consume hardware voraciously In a complementary way, highly parallel systems have structured properties that make the application of VLSI look very promising Moreover the remarkable advances of VLSI circuitry has lowered the implementation costs for large array processors to an acceptable level and has sparked research into the design of algorithms suitable for direct hardware implementation

Two popular special-purpose VLSI array architectures are systolic and wavefront arrays, which boast of massive concurrency derived from pipeline processing or parallel processing or both The concept of systolic architecture was introduced by H T Kung and C E Leiserson for VLSI implementation of some matrix operations [7] A systolic system can be defined as [3] - *a graph G=(V,E) of n interconnected PEs where the vertices represents the PEs and the directed edges represents the interconnection between the PEs The PEs operate synchronously controlled by a common clock An additional constraint was imposed*

that with the exception of the host all the PEs in V are to be Moore machines

(From here onwards the term PEs and cell will be used interchangeably)

A systolic system consists of set of interconnected cells, each capable of performing some simple operations Information flows between the cells in a pipelined fashion and communication with the outside world occurs only at the boundary cells The basic principle of a systolic architecture is as shown in Figure 1 1 By replacing a single PE with an array of PEs or cells, a higher computation throughput can be achieved without increasing the memory bandwidth The main idea of this approach is to ensure that once a data item is brought out from the memory it can be used effectively at each cell it passes while being "pumped" from cell to cell along the array This is possible for a wide class of compute-bound problems where multiple operations are performed on each data in a repetitive manner

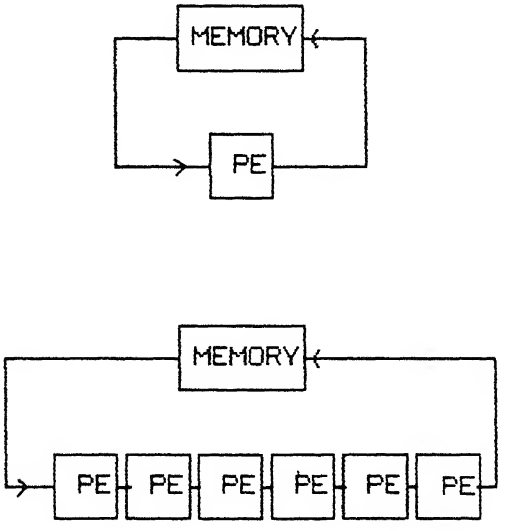


Figure 1 1 Systolic architecture principle

The basic features of systolic array are [20] -

i) Modularity and regularity - Systolic arrays are designed for solving a particular class of problem, and the array can be extended simply by adding new cells to handle larger size problems Application area can also be widened by adding new cells of different capabilities to meet additional functional requirements of the new problem The extra constraint of excluding the Mealy machines from the array guarantee that the number of PEs can be increased independent of the clock period

ii) Synchrony A single clock is used throughout the system to control the states of all Moore machines, i e PEs, in the system All the PEs perform either same or different operation on different data rhythmically and pass the data and partial results through the array

iii) Pipelineability The systolic array exhibits a linear rate pipelineability i e it achieves O(M) speedup, in terms of processing rate, where M is the number of PEs or cells in the array

iv) Spatial locality and temporal locality The array manifests a locally communicative interconnection structure, so that there is a high probability that the data needed in the near future may be located close to the data currently in use i e spatial locality Temporal locality is evident in a systolic array because the data keeps moving through the array in a systematic manner and there is at least one unit-time delay, so that data can be transferred from one cell to another

The prominent features of systolic arrays are the PEs and the regular interconnection between the PEs These features can be implemented either in software or with general-purpose DSP microprocessors or using specialized hardware The required level of granularity should be the main consideration for using any of these options

For some low-precision digital and image processing applications, it is advisable to use very simple processing primitives A good example of a commercial VLSI chip is NCR's Geometric Arithmetic Parallel Processor or GAPP Many DSP applications require the PEs to include more complex primitives Examples of commercial chips with a large granularity are INMOS's Transputer, NEC's data flow chip μpd7281, programmable DSP chips like ADSP-2100, TMS320, etc Some of the DSP algorithms require specialized hardware like floating-point ALU and multiplier, high speed RAMs, fast coefficient table addressing, etc To meet such requirement dedicated chips called DSP building blocks can be used

The concept of systolic architecture has been studied extensively over the last few years Some of the systems built to put this concept to use includes

i) The Warp [15] - designed by Carnegie Mellon University and its industrial partners, is used for signal and image processing tasks, low-level vision processing and scientific computing This machine has three main units the *array unit*, which consists of 10 or more identical cells, each with a throughput of 10 MFLOPS, connected in a linear 1-D array, the *interface unit*, which handles the communication between the host and the array, and the *host*, for supplying data to the array and to execute that part of the program which is not mapped onto the array unit

ii) The Systolic\Cellular System [8] - designed at Hughes Research Laboratories, and is used for large classes of linear algebraic and cellular operations that are used in signal processing applications It consists of a 16 X 16 mesh-connected array processor, a controller and dual-ported memory between the array and the host Each computational unit in the array is a custom built VLSI processor having 32 bit fixed-point dual-bus processor with a bit slice structure The maximum system performance is in the neighborhood of 450 MFLOPS

iii) Matrix-1 [8] - Designed by Saxpy Computer Corp for matrix

operations like matrix eigenvalue, singular value decompositions, matrix multiplication, Cholesky decomposition and QR factorization The system consists of - i) *system controller*, a DEC VAX, ii) *matrix processor*, a linear array of up to 32 pipelined floating-point processors that have systolic and global interconnections, iii) *system memory*, that stores data for the matrix processor, iv) *the mass storage system*, for high speed data-storage peripherals All these blocks are interconnected by the *Saxpy interconnect* The peak performance of Matrix-1 can reach up to 1000 MFLOPS and computation rate in excess of 900 MFLOPS have been achieved

1 6 OBJECTIVES AND ORGANIZATION OF THE THESIS

A Systolic Array Signal Processer (SASP) was defined at a system level by Nemawarkar [19] The SASP system consists of a linear array of identical cells, an interface unit and a host The objective of this thesis is to design and test a powerful and flexible cell for use in SASP

The systolic cell to be designed should have the following features -

i) It should be programmable

ii) The cell architecture should employ mutiple pipelined functional units

iii) The cell architecture should be optimized for a variety of algorithms encountered in signal processing algorithms

iv) The data format(s) to be used should have wide dynamic range

The thesis is arranged into the following chapters

With the brief introduction to signal processing techniques and the systolic architecture in particular in this chapter, Chapter 2 describes the architecture of the SASP system

In Chapter 3 we discuss the architectural considerations that were taken into account for deciding the cell architecture Also included in this chapter is a survey of various DSP building blocks available commercially

The design and implementation of the cell based on the cell architecture derived in Chapter 3 is explained in Chapter 4

Chapter 5 describes the instruction set and the software utilities developed for programming the cell This chapter concludes with an example of matrix-matrix multiplication

Finally in Chapter 6 we conclude the thesis with a note on the current implementation of the cell and suggestions for enhancement for the same

2. SYSTEM ARCHITECTURE

Systolic arrays are algorithmically specialized architectures Algorithms dictate the layout, the interconnection pattern among the cells and the structural features like pipelining of data and partial results Systolic systems are used as attached processors to a general-purpose computer and therefore should be able to adapt itself to different algorithms This chapter lists different types of systolic array structures, followed by the SASP (Systolic Array Signal Processor) system architecture [19]

2 1 SYSTOLIC ARRAY STRUCTURES

Over the last few years many researchers have tried to map different algorithms onto a systolic array Fortes et al have compared many existing methods for constructing a systolic array [11] CAD software packages have also been developed which can generate systolic array structure for a given algorithm Such structures can be classified into five broad categories They are

i) one-dimensional linearly connected array

ii) two-dimensional mesh connected array

iii) two-dimensional hexagonally connected array

iv) binary tree connected array

v) triangular array

Figure 2 1 shows various systolic structures and Table 2 1 lists some of the computations which map well onto these arrays

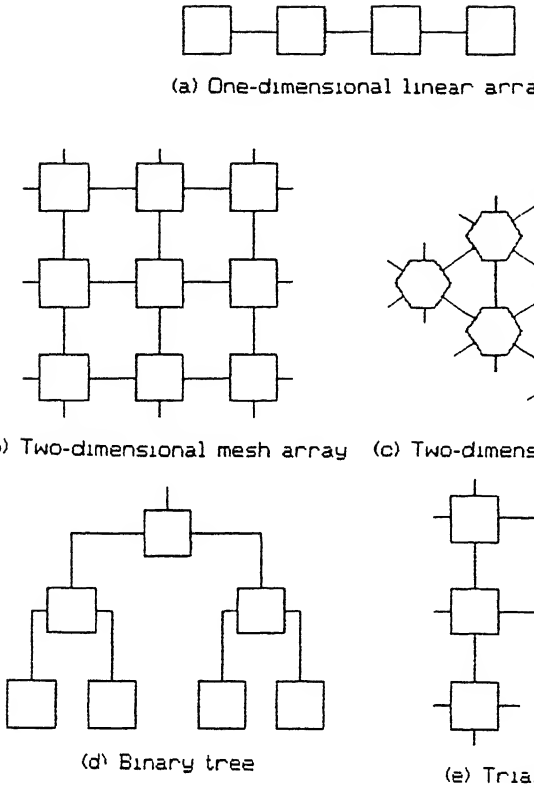


Figure 2.1 Various systolic array structures

Table 2.1 Systolic array structures [14]

Array structure	Applications
One-dimensional linear array	FIR filter, 1D and 2D convolution, discrete Fourier transform, solution of triangular linear systems
Two-dimensional mesh connected array	Graph algorithms, transitive closure, minimum spanning tree, shortest path, dynamic programming 2D convolution
Two-dimensional hexagonal array	Matrix multiplication, LU-decomposition, Gaussian elimination, QR decomposition
Binary tree	Searching algorithms, queries on nearest neighbor rank, etc, parallel function evaluation, recurrence evaluation
Triangular array	Formal language recognition QR decomposition

In Section 1.1 we had refered to some of the computations encountered in modern signal processing techniques. Most of these computations can be categorized into two distinct operations namely, i) matrix-vector multiplication and ii) matrix-matrix multiplication. Matrix-vector multiplication algorithms can be implemented very easily on a one-dimensional linearly connected array. Similarly matrix-matrix multiplication can be mapped on a two-dimensional hexagonal array. We now consider the time complexity of these two arrays for matrix-vector and matrix-matrix multiplication.

Consider a one-dimensional linear systolic array containing L PEs (Figure 2.2a). We will use cut-and-pile mapping (see Appendix 1) to map a large size problem onto a fixed size array. The objective is to compute the vector Y=AX+B, where A is an N-by-M matrix, and X and B are vectors of dimensions M and N respectively. The time taken, in terms of number of clock cycles, to compute all the elements of Y is $T \geq (2NM/L + 2L - 3)$ [13].

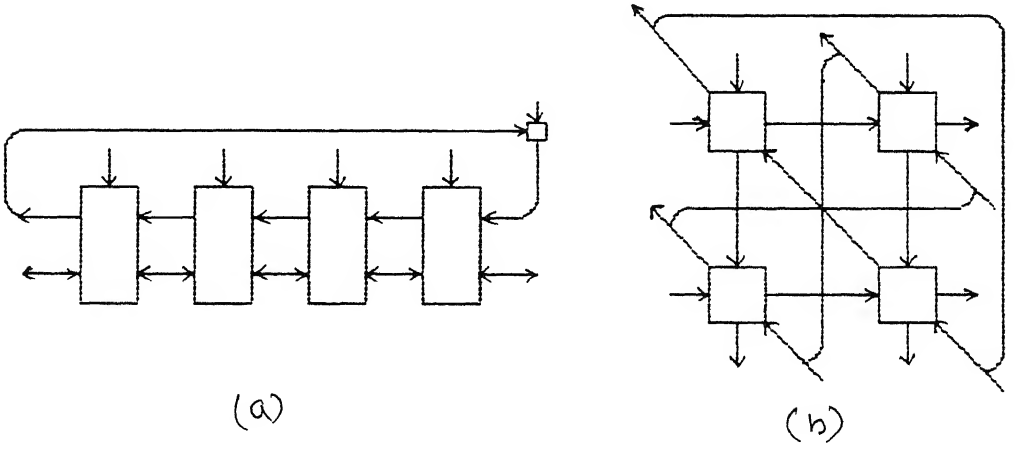


Figure 2.2 Linear and hexagonal array with cut-and-pile mapping

Now consider the computation E=FG+H to be performed on a L-by-L hexagonally connected PE array (Figure 2.2b). F, G and H are matrices of size M-by-N, N-by-P and M-by-P respectivley. Once again using cut-and-pile method to map a large size problem onto a array of fixed size, the total time, in terms of number of clock cycles, to compute all the elements of matrix E is $(3MPN/L^2 + 3MP/L + L)$ [13].

To compare these two structures, we define a term "array utilization" as $U=T_1/nT_n$, where n is the number of PEs in the array, T_1 is the number of clock cycles needed to solve the problem with only one PE and T_n is the number of clock cycles needed to solve it on the systolic array with n PEs. The ideal value of utilization factor for an array is unity. The utilization factor of the one-dimensional array approaches 1/2 for large values of N and M. Similarly the utilization factor for the two-dimensional hexagonal array approaches 1/3 when $NMP \gg L^3$.

From the above discussion we can conclude that one-dimensional linear array would be more suitable for signal and image processing applications. Moreover from Figure 2.2 one can see that the interconnections between PEs in a two-dimensional hexagonal array is more complicated than the interconnection in a one-dimensional linear array. Based on the above considerations we settle for the one-dimensional linear array for the SASP system.

2.2 SASP SYSTEM ARCHITECTURE

The SASP architecture is similar to the Warp system developed at Carnegie Mellon University [15]. The SASP machine will work as an attached processor to a general purpose host, a PC/XT in our case. The system consists of four major parts: the host, the interface unit (IFU), the processor array and the intercell communication channel (see Figure 2.3).

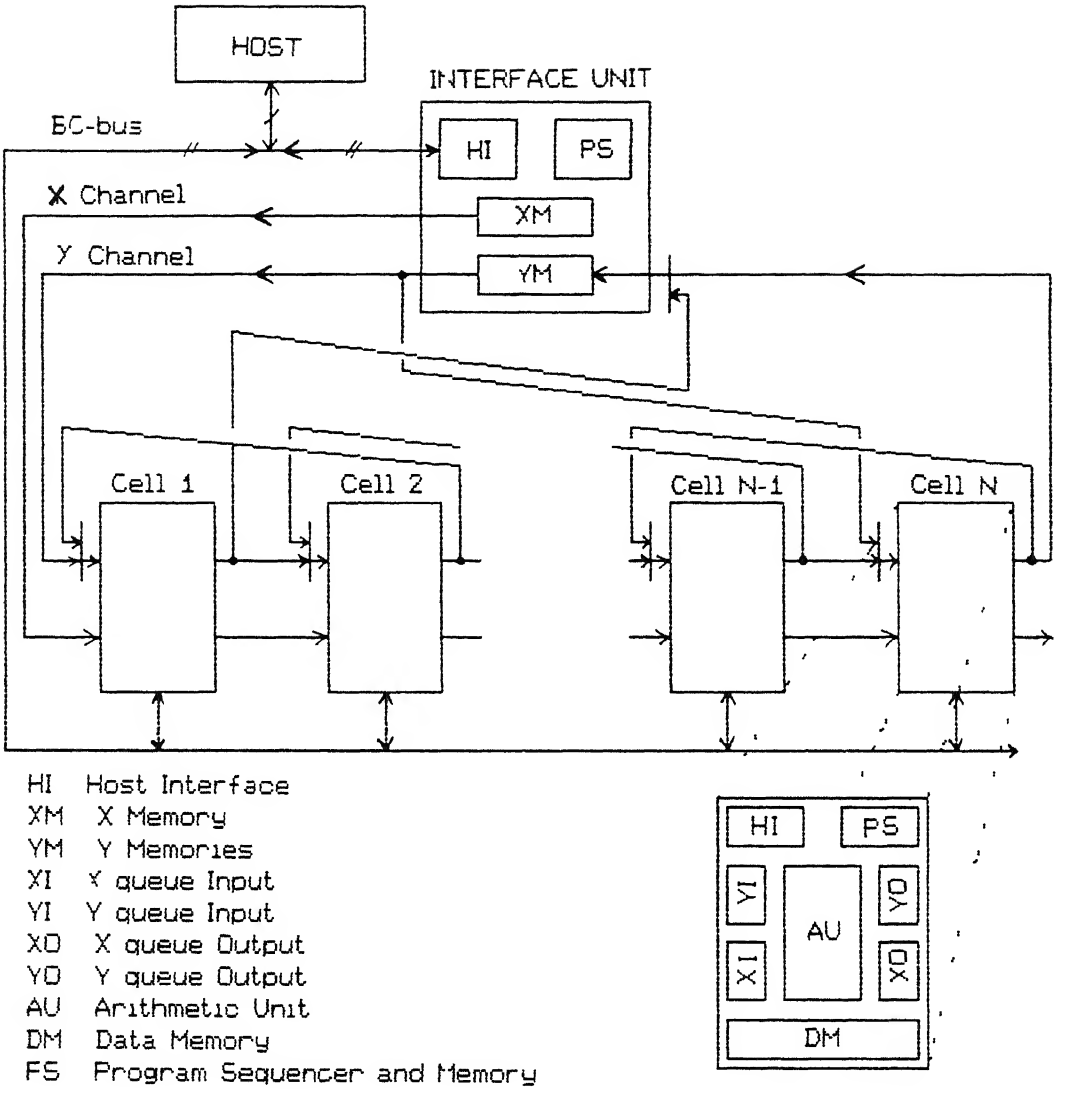


Figure 2.3 SASP system architecture

1) **Host** The host supervises the operation of the array by supplying data to the processor array and receiving the results back from the array. It is also used to download the programs into the cells. Further, the host may also perform those part of the computations that cannot be efficiently mapped onto the array. Any decision making during the execution of a problem is also taken by the host.

ii) **Interface Unit (IFU)** The systolic array requires the input data to the array be fed at a rate as determined by the algorithm to be executed Similarly the rate at which the array outputs the results is also specific to a problem During execution of large size problems on fixed small size array, intermediate results are generated, which will have to be fed back again into the processor array until the final output is available All these complex data handling cannot be done by the host, reason being the communication bandwidth (in terms of number of bytes/sec) of host cannot match the bandwidth requirement of the array Further, it is desirable that the host and the array operate independently

This then requires a special unit (called the IFU) which can match the bandwidth of the host and the processor array Apart from matching the bandwidth the IFU has the following functions - i) to route data according to the array configuration i e , forward or backward, ii) receiving intermediate results and looping them back into the array, iii) receiving and storing the final results, and iv) to provide global clock and synchronizing signals to the processor array

iii) **Processor Array** This is the computing unit of the whole system It consists of identical cells, connected to two adjacent cells on either side Each of the cell is a programmable horizontal microcoded system, with its own program sequencer, address generator, floating-point multiplier and ALU and data memory All the cells and the IFU are connected to a broadcast bus (BC-bus) which is used only by the host to communicate with the IFU and the cells The main usage of this bus is to download the microprogram to the IFU and the cells, load initial data in the data memories, and read data/results from the data memory in case when the cell interrupts the host due to abnormal program termination Section 2 4 describes the BC-bus in more detail

iv) **Intercell communication channel** In SASP intercell communication takes place on two communication channels called the X and Y channel These channels are to be implemented using FIFOs The details of these channels are as follows

X channel This unidirectional channel originates at the IFU and ends at the last cell, i e , the right-most cell Data which will be used for multiple computations is transmitted over this channel The X data passes through each cell unmodified, with each cell using the data on the X channel to perform some computation locally

Y channel This is a bi-directional channel, whose direction is controlled by the algorithm to be executed This channel forms a closed loop - originating at the IFU, traveling through the cells and finally terminating at the IFU Partial and final results moves on this channel This channel helps in implementing a large size problem on an array with limited number of cells

The communication channel also has hardware flow control When a cell tries to read from an empty queue of the neighbour cell, it is blocked (i e , the cell does nothing) until data arrives Similarly, when a cell tries to write into the neighbour cell queue which is full, the writing cell is blocked until, at least, one data is removed from the queue The blocking of a cell is transparent to the user program Except for the blocked cell(s), all other cell(s) in the array continue to operate normally

As each cell is to have its own address generator, it was felt that a separate address queue will not be necessary The address generator should be capable of generating linear address, address for circular buffer, bit-reversed address which are frequently encountered in signal-processing techniques

2.3 BROADCAST BUS

As stated earlier the broadcast bus is used only by the host to write into, or read from the memories of the cell The signals on this bus are as follows

1) Data bus This data bus is used by the host to transfer data to/from the cells

2) RESET* This is a software reset signal used to reset the sequencer in the cell This signal is also used to synchronize the execution of all the cells

3) WCS (Write Control Store) Assertion of this signal forces the sequencer (ADSP-1401) in a cell to execute a WCS instruction After executing this instruction the sequencer addresses the microcode memory sequentially, starting from 0000, so that the host can program the cell

4) Cell address The four bit address is used to select a cell or the IFU, with which the host wants to communicate Out of the sixteen addresses one address is reserved for broadcast purpose

The address assignments are as follows -

0	interface unit (cell 0)
1 to 13	one of the fourteen SASP cells
14	broadcast address i e , all the cells are selected
15	no cell is selected

5) Clk This is the global clock supplied by the IFU and is used by all the cells in the system to time its activities

6) HWR* This signal is supplied by the host and is used to latch data from the host in the microcode memory and the data memory

7) HRD* This signal is supplied by the host and is used to read data from the microcode memory and the data memory

2.4 PROTOTYPE SASP

Work on the SASP was initiated in 1988 by Nemawarkar [19] where he had defined the system architecture for SASP. Based on this design a prototype of the IFU for SASP was designed by Samit [21] and Usman [17]. The salient features of this unit are - i) it is microprogrammable, and has ii) a X memory which holds the data to be passed on to the processor array, iii) a YA memory which holds the initial values and final results received from the processor array, iv) a YB memory which holds the intermediate results output by the processor array. This unit has 8 bit data path throughout.

A small cell with limited capability was also developed by them to test the working of their IFU. The cell was implemented using a 8 bit multiply-and-accumulate chip (ADSP-1008A). This cell had a local memory of 8192 bytes to store the local data. The X and Y communication channels in this cell was implemented using a simple latch. The IFU was able to transfer data to/from its X, YA and YB memories to the latches in the cell. The IFU along with the cell was able to run programs like matrix-matrix multiplication, linear convolution and AR filtering. Further, Usman has also discussed a design for 32 bit cell with enhanced capabilities.

In parallel with the design of a 32 bit cell, as a part of this thesis, a 32 bit IFU has been designed by Shera [23] to complement this new cell.

Summarizing, the linear array structure of SASP simplifies intercell communication, without reducing the communication bandwidth between the adjacent cells. New cells can be added, because of their modularity. For an array with large number of cells, global synchronization can be a problem. Keeping this in view, run time flow control was incorporated in the architecture so that switch over to wavefront architecture is feasible.

3. CELL ARCHITECTURE

The SASP cell should be able to efficiently perform the computationally intensive portion of signal processing This requirement justifies the inclusion of high-performance hardware in the design of the cell The considerations to be taken into account before deciding the cell architecture are taken up in the next section This is followed by a brief survey of the DSP building blocks available commercially The final configuration of the cell is developed in the last section

3 1 ARCHITECTURAL CONSIDERATIONS

Computations encountered in signal processing are regular and repetitive For these applications the cells in SASP can be uesd either in pipeline or parallel mode Pipeline mode of computation require a high intercell communication bandwidth Parallel mode of operation requires the cell to be more powerful and capable of operating independently For the cell to operate in either of these two modes the following considerations were taken into account

3 1 1 DSP System Alternatives

Presently available hardware offers two alternatives for the design of DSP systems They are - i) DSP microprocessors and ii) microprogrammable basic building blocks

A DSP microprocessor contains all the computational elements on a single chip and has separate bus structures for program and data memories (the so called Harvard architecture) The resulting improvement in memory bandwidth (i e , number of bytes transferred per second) is still not sufficient for real-time

or high-speed signal processing Operations like filtering, convolution, etc, require multiple operands for each computation and the instruction sets of the currently available DSP microprocessors cannot address multiple operands simultaneously because of their limited number of data paths to and from the external world[1] Further, these microprocessors do not allow the user to use all the inherent parallelism available in the hardware For example, a STORE to memory could logically be accomplished at the same time as a JUMP to a branch routine This restriction is mainly due to the vertical microprogramming used in the control unit of these microprocessors In vertical microprogramming the instruction bits are fully utilized with each instruction specifying a single discrete operation

A microcode-based system design using basic DSP building blocks offers a high degree of functional parallelism and hence a higher throughput The main features of such a system are - i) multiple functional units, ii) multiple interconnections and iii) multi-operation instructions The system can be tailored to meet the specifications by having multiple functional units and data paths not available otherwise Further, microprogramming gives direct access to the internal parallelism of the hardware The designer can either employ a complete horizontal microprogramming, where the instruction bits are not fully utilized and several operations can be initiated simultaneously, or use a combination of horizontal and vertical microprogramming depending upon the system architecture

For a high degree of functional parallelism the cell should include the following functional blocks -

---

[1]This is true for only those DSP microprocessors which employ a pseudo Harvard architecture, for example TMS32010

* ALU capable of addition, subtraction and logic operations
* Multiplier
* Data-memories
* One-to-one connection between different blocks i e , crossbar switch
* Address generators for data-memories
* Program sequencer

3 1.2 Cell Optimization For Signal Processing Applications

As mentioned in Section 2 1 most of the signal processing algorithms involve matrix-vector or matrix-matrix multiplications The main arithmetic operation encountered in signal processing is the *inner product* or the Multiply-and-ACcumulate (MAC) operation [1]

This operation is so common that we can as well provide a MAC unit without requiring an intervening store and load of the product to and from a high-speed register Since our system is not meant to be fine-tuned for a particular operation, it is better to have a separate multiplier and an ALU with a provision to load the multiplier output directly into the ALU Moreover providing just a MAC will reduce the functional parallelism as we cannot initiate a separate multiplication and addition simultaneously

To properly integrate the SASP in a variety of host environment, the number representation system used both by the cell and the host should match exactly Further it is desirable that the number system has a wide dynamic range Both these features are available in the IEEE 754 standard for 32 bit single-precision and 64 bit double-precision floating-point number system [9]

Division operation is not encountered frequently in signal processing applications Therefore instead of providing a separate unit, division can be programmed in software by a small look-up table and a Taylor series expansion

involving multiplications and additions only [4]

3.1.3 Vector And Scalar Processing Capabilities

Supercomputers like Cray-1, VP-100, etc., provide one set of hardware to perform vector operation and another set to perform scalar operations. Significant saving in hardware can be achieved if the same hardware can be used for scalar as well as vector processing. Vector processing is repetitive in nature and data is stored in consecutive memory locations to facilitate addressing of the data. On the other hand occurance of scalar operations are usually more random and the data arrangement in memory may not be sequential. To strike a balance between vector and scalar computations without degrading either of them it is necessary to use (a) functional units (ALU and multiplier) with a minimum number of pipeline stages (two or three stages of pipelining gives an optimal balance between the scalar and vector computation [1]), and (b) separate address generator capable of fast memory access, with little programming overhead, for a variety of access modes like sequential, circular buffer, bit-reversed and non-sequential.

3.1.4 Data Memories

To initiate a triadic operation like a=b*c+d every clock cycle, we must be able to supply three operands to and receive one result from the functional units. As only one operand resides in the cell and the other two operands will be received from the neighboring cells, there will be memory contention as a cell tries to access the memory of its adjacent cell for data transaction.

A simple solution to the above problem will be to provide two dual-ported memories in each cell, one for $cell_{i-1}$ and another for $cell_{i+1}$. There are two main drawbacks of this approach. Firstly, there is no way of detecting the

fact that $cell_{i-1}$ or $cell_{i+1}$ has overwritten a memory location before $cell_i$ could use the previous data Secondly, address lines will have to be routed from $cell_{i+1}$ and $cell_{i-1}$ to $cell_i$

Noting the fact that the access to the data received by $cell_i$, from its neighbor is going to be always sequential [20,21], the above said drawbacks can be avoided by replacing the dual-ported memory with a FIFO Usage of FIFO obviates the need for the address bus and also blocks any attempt to overwrite a data before $cell_i$ can read it [16]

3 1 5 Functional Unit Interconnections

Keeping the functional units like ALU and multiplier occupied in every clock cycle requires a high bandwidth interconnection between the memory and the functional units Interconnection schemes range from a single bus, where only one item can be transferred at a time, to a fully connected crossbar switch where all possible connections can be made simultaneously Crossbar connection has the highest efficiency and offers the highest bandwidth and therefore it is decided to use a full crossbar to interconnect the memory and the FIFOs to the ALU and the multiplier The crossbar can be implemented using PALs or multiplexers and demultiplexers

3 1 6 Register File

A register file increases the communication bandwidth by using multiport memory Inclusion of a register file facilitates the following

i) Flexible data flow between the multiplier and the ALU

ii) To feedback the output of ALU to its input with some delay, useful in mac operation

iii) As the data memory can be used either in read or write mode in a

single clock cycle, the register file could be used to hold those data which will be written to the data memory at a later clock cycle

ii) *Gather*[2] *and scatter* operation for sparse matrix without using expensive hardware like *bit vector* as in Cyber-205 [14]

The cell architecture shown in Figure 3 1 is based on the above mentioned considerations and the system architecture as described in chapter 2

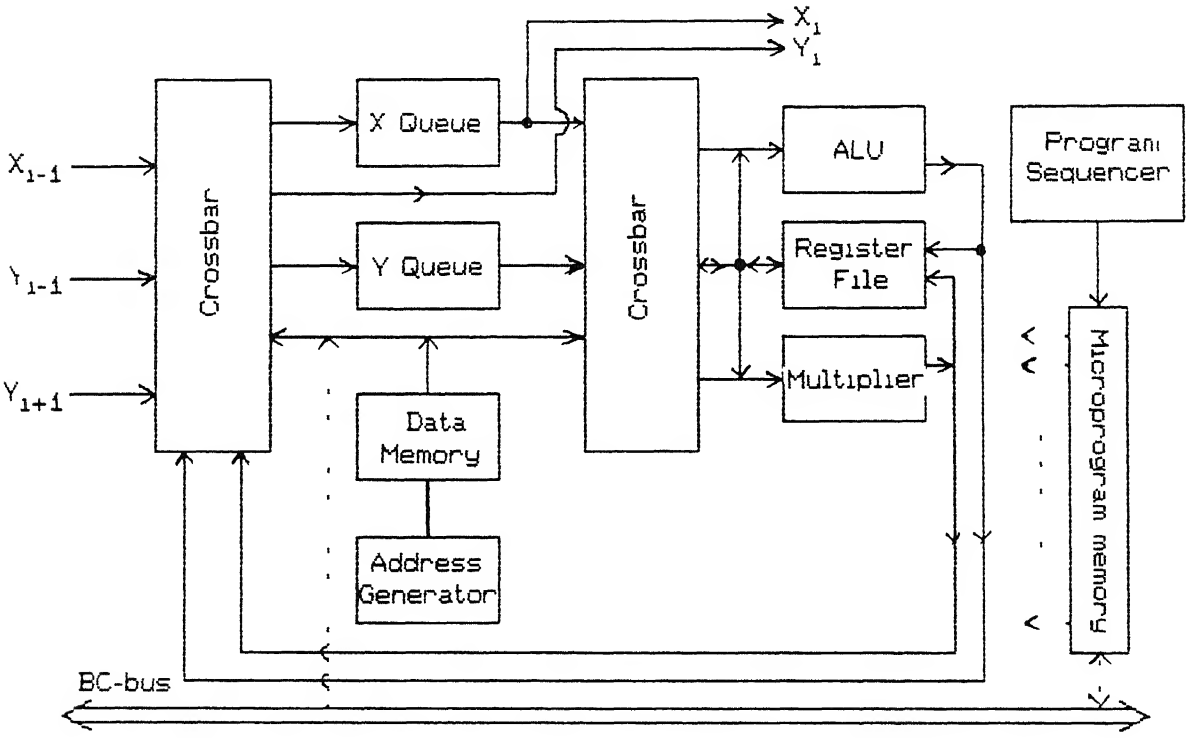


Figure 3 1 Cell architecture

[2]A sparse matrix contains few non-zero elements To reduce the computation time for such matrices the non-zero elements are *gathered* and a new array is formed This array can be stored in the register file and pipelined through the functional unit at a maximum speed The result of these computations can be stored back in the register file Finally these results are *scattered* in the data memory in appropriate locations of the original array

3.2 DSP BUILDING BLOCKS

Advances in VLSI have made it possible to squeeze in more and more hardware in a single chip This had lead to the design of different efficient functional units that can be used for real time digital signal processing In this section we will consider some of these functional units

3.2.1 Program Sequencer

A program sequencer's main task is to sequence the microprogram by providing appropriate address for the next instruction in every clock cycle The sequencer has several other functions and are as listed below

i) Handle normal program flow, incrementing the program counter by one in each cycle

ii) Keep track of subroutine addressing, and manage the return address stack

iii) Manage loops with little overhead, using on-chip loop counter(s)

iv) Service interrupts from external devices

v) Branch to an instruction, either conditionally or unconditionally, using a direct address or an address offset read through an external port (Most sequencers provide two-way and three-way jumps within a single instruction which can speed execution and ease programming by eliminating the need to code explicit loop testing)

Commercially available program sequencers includes ADSP-1401, ADSP-1402, Am2910, IDT39C410, IDT49C410 and IDT49C411 IDT39C410 is pin compatible with Am2910 but comsumes less power Both of them provide 12 bit address for microprogram memory A 33-deep LIFO stack provides microprogram subroutine linkage and looping capability A 12 bit counter is also provided for loop iteration and repeating instructions IDT49C410 is also functionally equivalent to

Am2910 but has 16 bit wide microprogram memory address bus and a 16 bit loop counter The Am2910 family lacks relative jump instruction and has no provision for hardware interrupts The IDT49C411 has a 20 bit wide address bus, a 20 bit loop counter, three independent 64-deep stack and can handle eight prioritized interrupts ADSP 1401 is much more versatile than the Am2910 family It has 16 bit address bus, four independent loop or event counter, ten prioritized interrupts and a 64 word stack with three stack pointers It can also execute relative jump and subroutine call ADSP 1402 is functionally equivalent to ADSP 1401 and has some additional glue logic built in it which otherwise will have to be provided by the designer

3.2.2 Address Generator

Fast and flexible addressing of data is of import in digital signal processing application Data array scanning is one of the most common address manipulation in DSP Data array scanning can be sequential or decimated i e every second, fourth, etc , location may have to be addressed Further, most DSP lookup tables are circular buffers The address generator should be able to reset the pointer to the start address of the buffer when the pointer is equal to some preset boundary value For highly structured algorithms such as FFT, the data access may be decimated in time or in frequency and the address needs to be bit reversed These requirements calls for a specialized address generator capable of performing the above mentioned operations

Of all the commercially available address generators ADSP-1410 is a general-purpose address generator which has all the above said features Am2901 is a 4-bit slice which needs to be cascaded to get a adequate addressing range This chip lacks circular buffer and bit-reversed addressing feature The IDT49C402 is a super set of Am2901 and is functionally equivalent to four Am2901

and one Am2902 Am29540 is a special-purpose address generator which can generate both data addresses and twiddle-factor addresses for either radix-2 or radix-4 butterflies, bit reversed input or output, and decimation either in time or in frequency

3.2.3 FIFOs

The first generation FIFOs were of the "register array" architecture and used "bucket brigade" arrangement to move data out of the FIFO [16] The second generation FIFOs use dual-ported static RAM in order to achieve a truly independent, asynchronous operation of the input and output The RAMs in the FIFO are internally addressed using two counters one for read and the other for write In addition, a bit is used for every FIFO word to designate which word has been written to but not yet read These FIFOs width and length can be increased by juxtaposing and cascading them with few extra logic FIFOs are available in various size from different manufacturers Listed below are some of them

CY7C412/IDT7201A	- 512 X 9 bits
CY7C424/IDT7202A/IDT72021	- 1024 X 9 bits
CY7C429/IDT7203	- 2048 X 9 bits
IDT7204/72041	- 4096 X 9 bits

3.2.4 Floating-point ALU and Multipliers

There is a heavy dependence on floating-point arithmetic for signal processing applications because of a potentially large dynamic range of values A discrete solution to a 32 bit floating-point processor can be at least one board of SSI and MSI circuits So it is preferrable that an IC or an IC set that implements the IEEE standard be used Further as DSP applications require high speed operation, the multiplier should use array multiplier and the multiplication

operation should not span more than two or three stages of pipeline Table 3 1 and 3 2 contains a list of available floating-point chipsets along with their salient features Some of these floating-point chip sets provide two mode of operations (i) pipelined and (ii) "flow-through" (i e , the pipelined registers made transparent) The tradeoff in using pipelining is that new values are loaded every clock cycles and one result is available every clock cycle, once the pipe is full In the flow-through situation, one does not present any new operands to the inputs during the duration of the operating time

3.2.5 Register File

Register files are multiport memories that provides high speed local storage for data and flexibility in data transfer from one port to another Register files can also be considered as data cache, for they are generally used

Table 3 1 32 bit floating-point multipliers

	Input port(s)	Output port	Operand types					Pipeline Stage(s)	MFLOPS	
			SP	DP	TC	US	MM		SP	DP
ADSP 3201	two	one	Y	---	Y	Y	---	one	10	---
ADSP 3210	one	one	Y	Y	Y	Y	---	one	20	5
ADSP 3212	two	one	Y	Y	Y	Y	---	one	20	20
IDT72264/WTL1264	two	one	Y	Y	---	---	---	one	16 6	8

Table 3 2 32 bit floating-point ALU

	Input port(s)	Output port	Operand types					Pipeline Stage(s)	MFLOPS	
			SP	DP	TC	US	MM		SP	DP
ADSP 3202	two	one	Y	---	Y	Y	---	one	10	---
ADSP 3220	two	one	Y	Y	Y	Y	---	one	10	10
ADSP 3222	two	one	Y	Y	Y	Y	---	one	20	10
IDT72265/WTL1265	two	one	Y	Y	---	---	---	three	16 6	16 6

SP (single-precision floating-point), DP (double-precision floating-point),
TC (twos-complement fixed-point), US (unsigned fixed-point),
MM (mixed mode integer), Y (yes)

to hold data that are used frequently or will be used in the near future i e, spatial locality A register file provides temporary data storage and simultaneous input and output on several pieces of data, thereby expanding the computational bandwidth of the system As said earlier register file is a static RAM surrounded by latches and control logic needed for simple system interface Each port has its own address latch and read/write signals and is permanently configured as read only port or as write only port However it is possible that one or two port can be configured as read and write port Access to the memory from all ports are time multiplexed and different ports are assigned different priorities Because of the plethora of data paths, many combinations of reads and writes are possible Commercially available register files include the ADSP-3128 and WTL 1066

3 3 FINALIZATION OF CELL ARCHITECTURE

From the foregoing survey of DSP building blocks we can see that Analog Devices provides a complete set of devices which are meant for high speed digital signal processing applications We have selected the following components for implementing the cell

ADSP-1401 - program sequencer

ADSP-1410 - address generator

ADSP-3210 - floating-point multiplier

ADSP-3220 - floating-point ALU

ADSP-3128 - five port register file

IDT7202A - 1024 X 9 bit FIFO

The salient features of these devices are given in Appendix 4

Although the configuration given in Figure 3 1 is capable of the basic operations needed by a cell some additional hardware like auxiliary memory and

appropriate choice of interconnection would make the cell operation even more flexible

3.3.1 Auxiliary Memory

As stated earlier it might be necessary that each cell be assigned the job of more than one cell when the problem size is greater than the number of cells in the array. Under such circumstances the cell might have to hold several partial results within itself. If we use data memory to hold partial results, there will be a heavy traffic between the data memory and the functional unit. Further as data memory can be used either for a read or a write operation during a single clock cycle the throughput will be affected. To hold such data with high temporal locality it was decided to add a small memory connected directly to the ALU and the multiplier without increasing the size of the crossbar. The auxiliary memory will also have a read-only portion for frequently used constants such as sine, cosine table for FFT and inverse table for division operation.

3.3.2 Simulating Multiple Cells

Two ways of mapping large size problems onto a fixed size array is given in Appendix 1. Both these methods require some extra hardware to facilitate the mapping and to preserve the properties of systolic array mentioned in Section 1.3.

Cut-and-pile mapping requires a link between the last cell and the first cell thereby forming a closed path. This closed path is easily provided by including the IFU in the loop. The IFU will receive data from the last cell and pass it to the first cell.

Coalescent mapping requires a loop back from the cell's output to it's input. This loop is required on the X and Y data link and can be easily

implemented by using multiplexers at the X and Y queue inputs

3.3.3 Register File Connection

The register file (ADSP-3128) that we have chosen has five ports. Port A and B are write only ports, port C and D are read only ports and port E is a bi-directional port. Each port is 16 bit wide. The ALU, multiplier and the register file can be connected together in a number of different configurations. After evaluating different configurations we have selected the configuration shown in Figure 3.2 for it offers the most flexible interconnection between the ALU, auxiliary memory, crossbar, multiplier and the register file.

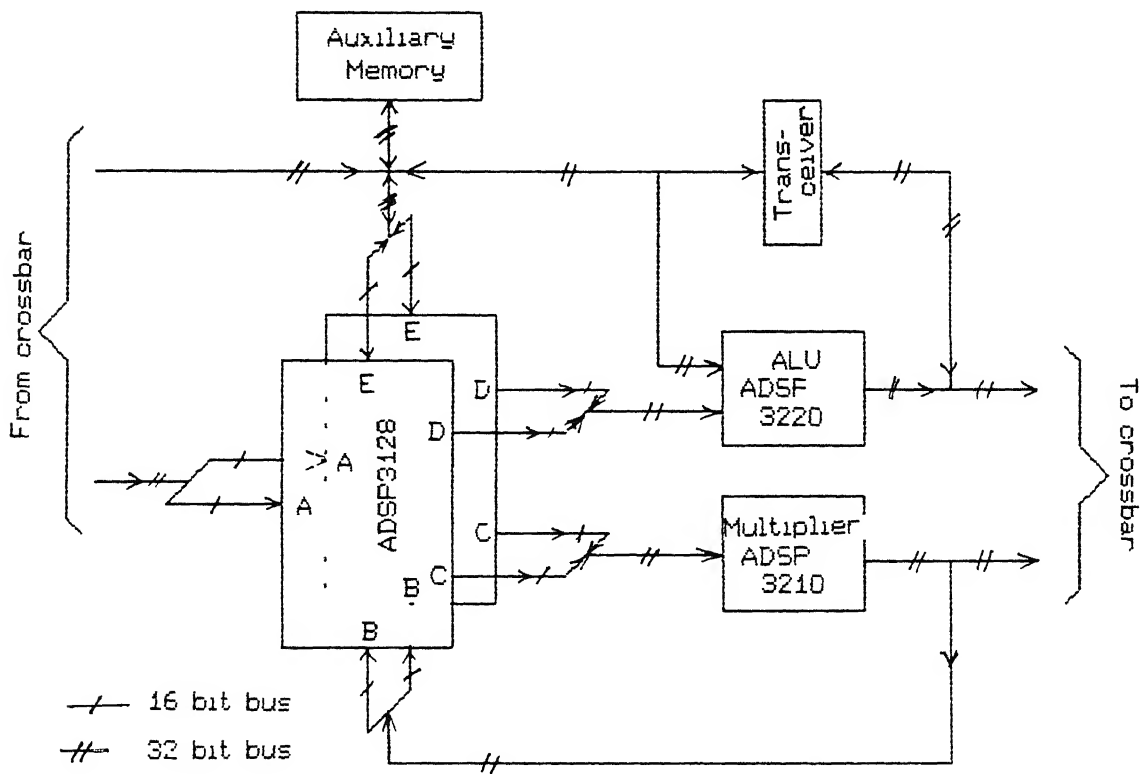


Figure 3.2 Register file, ALU, multiplier and auxiliary memory interconnection

3.4 FINAL CELL ARCHITECTURE

Augmenting the cell architecture derived in Section 3.1 with the refinements suggested in Section 3.3 the final cell structure is as shown in Figure 3.3. The cell uses 32 bit wide data paths throughout without any tradeoff between complexity and efficiency.

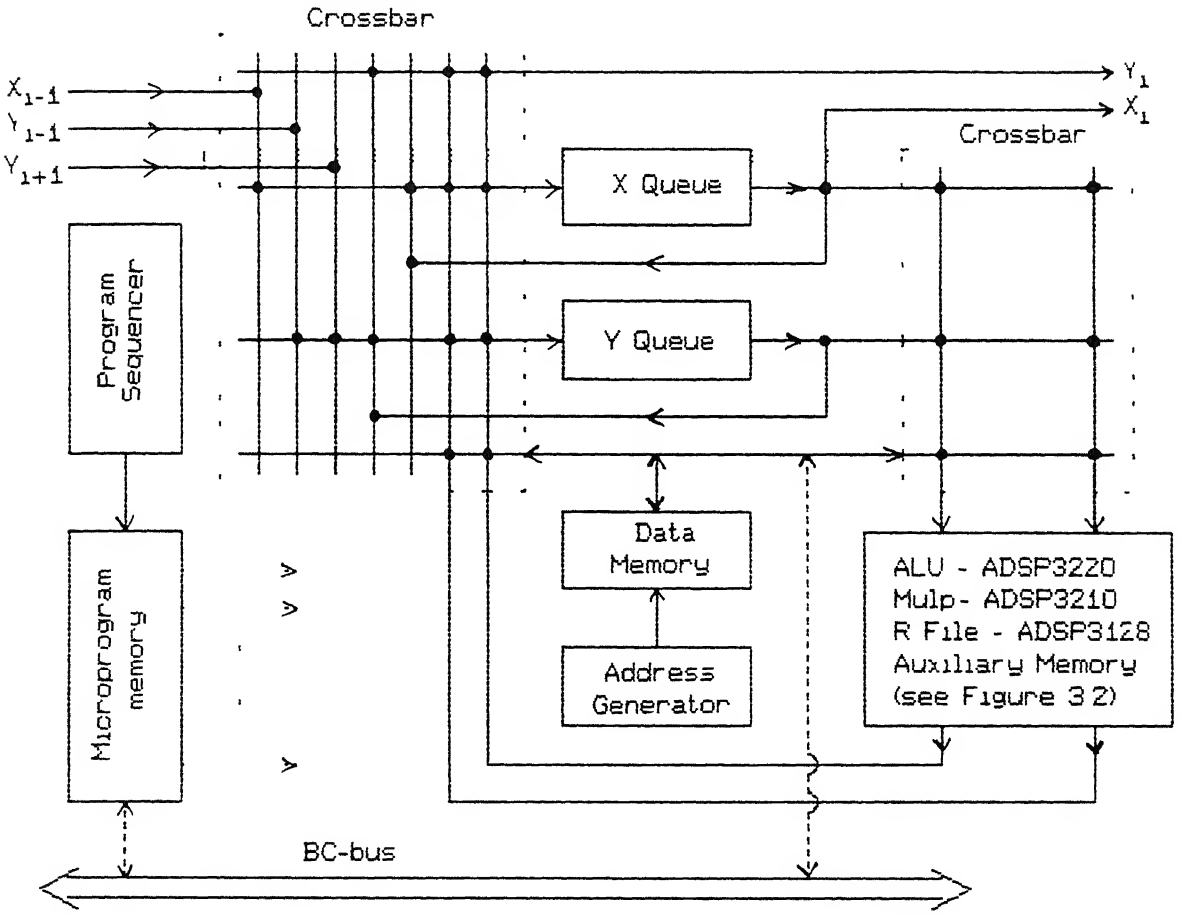


Figure 3.3 Final cell architecture

4. CELL DESIGN AND IMPLEMENTATION

The SASP cell implementation is spread over three different modules The first module called the microengine contains the program sequencer and microprogram memory The second module called the data cache includes the register file and auxiliary memory The third module called the number-crunching unit houses the crossbar circuitry, the data memory, the X and Y queues, the floating-point ALU and the multiplier Presently only the first and the third module has been implemented

4 1 MICROENGINE

A block diagram of the microengine is as shown in Figure 4 1 The microengine consists of a program sequencer (ADSP-1401), an address generator (ADSP-1410), host interface circuitry, two control registers, host read/write synchronizing circuitry and 128 bit wide microprogram memory This module is assembled on three PCBs

4 1 1 Host Interface

The cell is mapped onto the host's (PC\XT) I/O addressing space A prototype adapter card plugged into the PC\XT decodes the address lines A15 to A5 (Figure 4 2) The decoded address generates the signal EN* which acts as the cell select signal (In case of multiple cells this signal will be generated by decoding the cell address lines on the BC-bus) The EN* signal is ORed with CRW* (i e , IOR* ANDed with IOW*) to generate a new signal BEN* which is used to enable a transceiver (74LS245), which connects the PC data bus to the cell memories

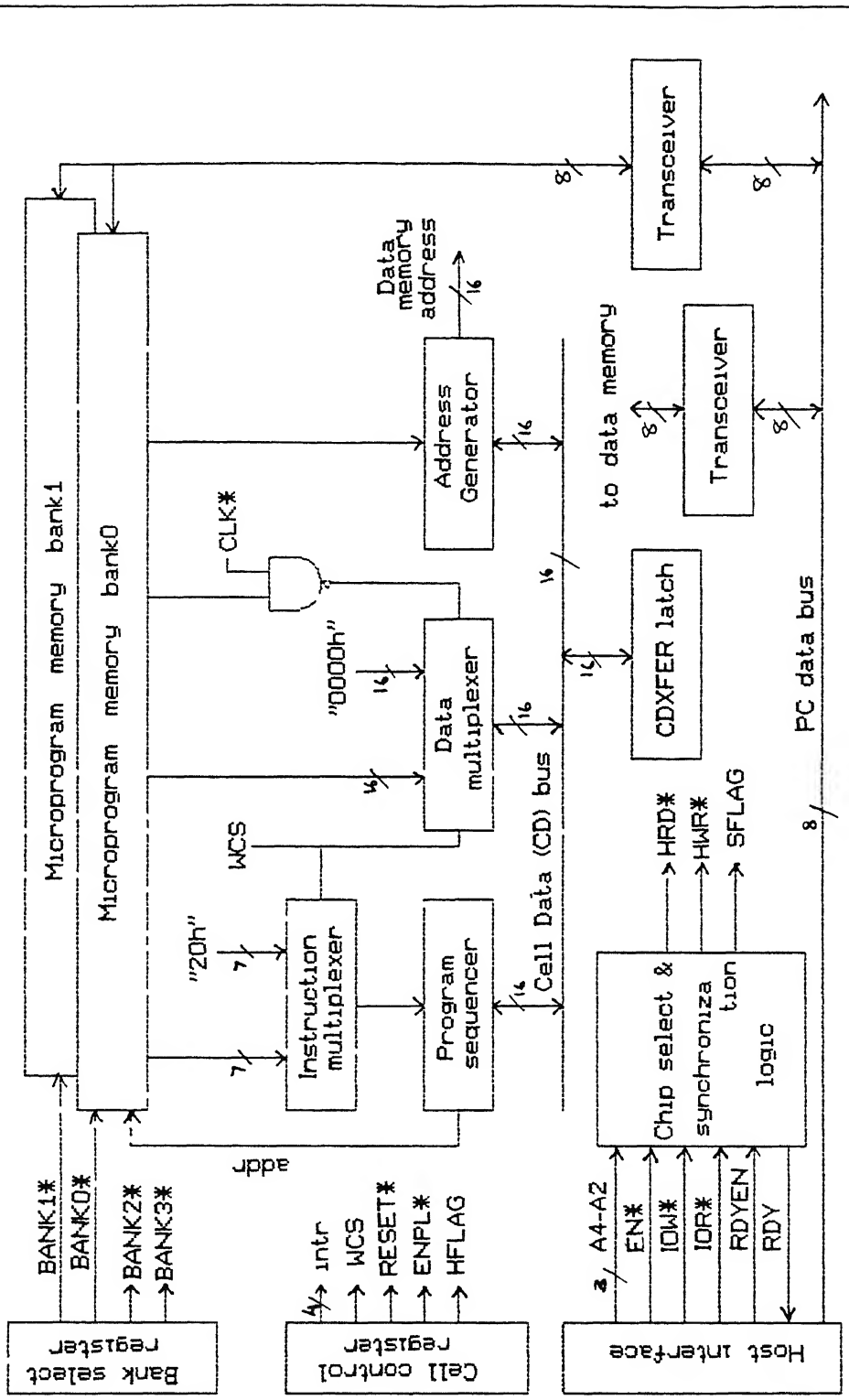


Figure 4.1 Microengine block diagram

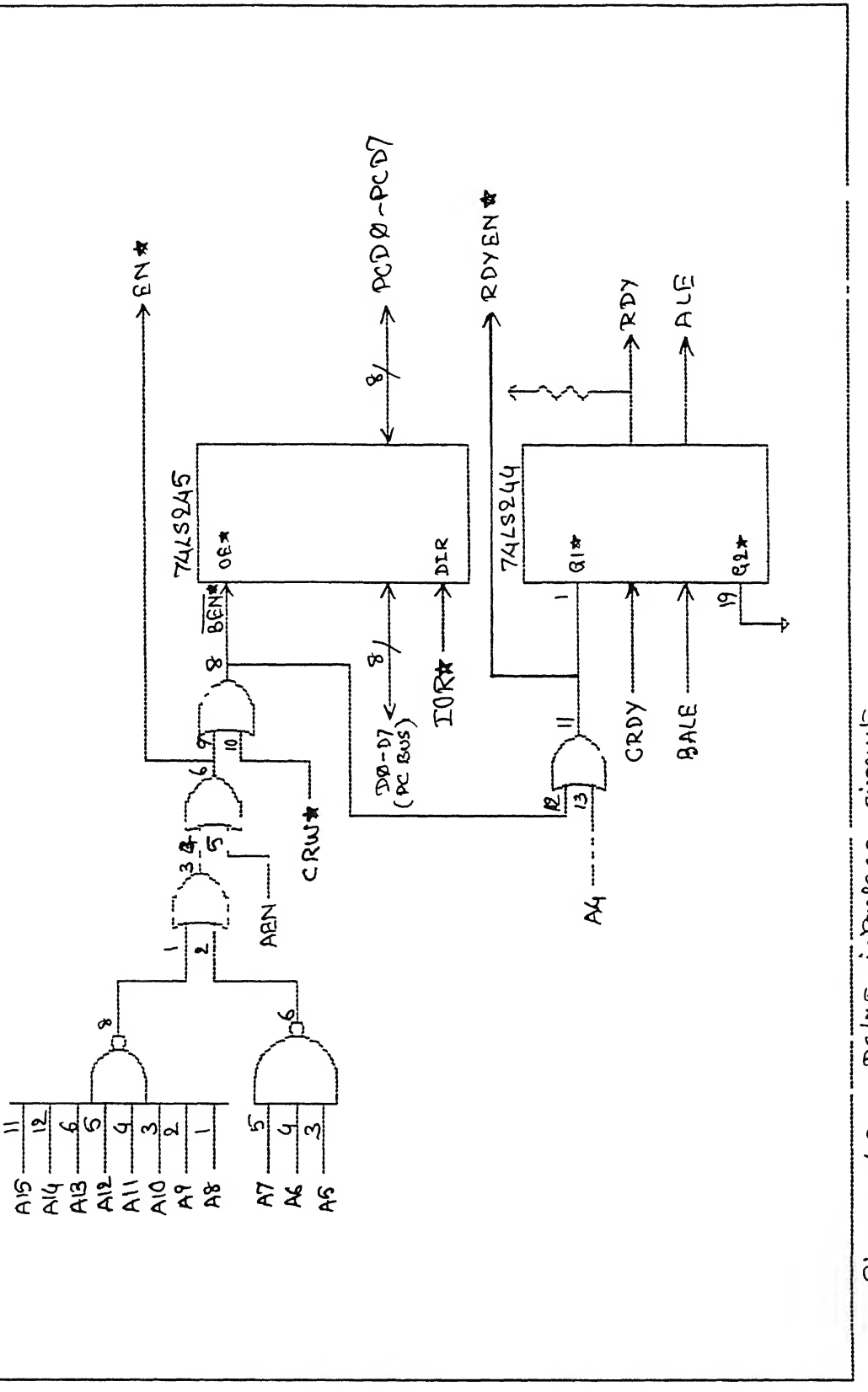


Figure 4.2 PC/XT interface circuit

A 3-to-8 line decoder (74LS138), located on the microengine card, decodes the address line A4 to A2 to generate chip select signal for different registers and transceivers This decoder is enabled by EN* signal The different chip-select signals and their usage are as listed below

Signal	Address (in hex)	Usage
CS0*	FFE0 to FFE3	Microcode memory read/write
CS1*	FFE4 to FFE7	Data memory read/write
CS2*	FFE8 to FFEB	not used
CS3*	FFEC to FFEF	not used
CS4*	FFF0 to FFF3	not used
CS5*	FFF4 to FFF7	Cell control register (write only)
CS6*	FFF8 to FFFB	Microprogram memory bank select register (write only)
CS7*	FFFC to FFFF	Cell status register (read only)

Reading from or writing to cell memories by the host requires synchronization between the host and the sequencer This then requires that wait states be introduced whenever the host accesses the cell memories The prototype card contains a buffer (74LS244) which is enabled to pass the CRDY signal from the microengine card to the PC bus This buffer is enabled by RD\EN* signal which is EN* ORed with A4 address bit, i e , wait states will be introduced for CS0*, CS1*, CS2* and CS3* For access to other registers this buffer is not enabled and it's output is pulled high through a pull-up resistor

4 1.2 Control Registers

There are two control registers in the cell (see Figure 4 3) They are - i) the cell control register and ii) the microprogram memory bank select register The cell control register is used by the host to put the sequencer in a known state and the bank select register is used to select different banks of the

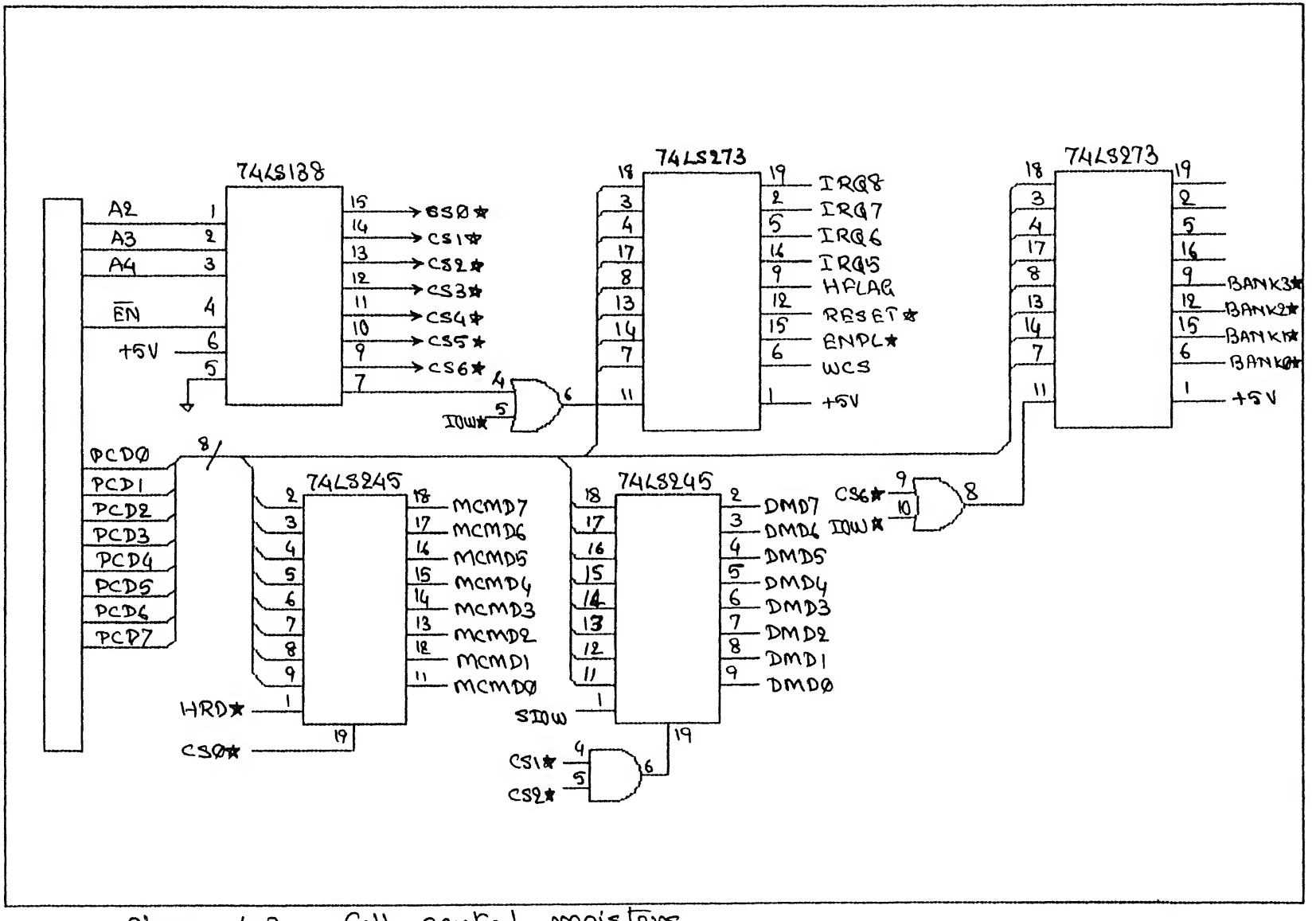


Figure 4 3 Cell control registers

microprogram memory while downloading the microprogram The signals associated with these registers and their description follows

i) Cell control register

Bit 0	WCS	Setting this bit to logic one causes the program sequencer instruction multiplexer to present a WCS instruction to the sequencer This bit should be set only when the host wants to download some microprogram
Bit 1	ENPL*	When this bit is set to logic zero, the pipeline register output is enabled, otherwise it is tri-stated
Bit 2	CRES*	When this bit is set to logic zero, the sequencer is held in reset state When set to logic one the sequencer starts executing from location 0000
Bit 3	HFLAG	This bit can be read by the program sequencer at it's FLAG input and can be set/reset by the host to change the program flow
Bit 4	IRQ5	Hardware interrupt from host to the sequencer
Bit 5	IRQ6	Hardware interrupt from host to the sequencer
Bit 6	IRQ7	Hardware interrupt from host to the sequencer
Bit 7	IRQ8	Hardware interrupt from host to the sequencer

ii) Microprogram memory bank select register

Bit 0	BANK0*	Select microprogram memory bank 0
Bit 1	BANK1*	Select microprogram memory bank 1
Bit 2	BANK2*	Select microprogram memory bank 2
Bit 3	BANK3*	Select microprogram memory bank 3
Bit 4		Not used
Bit 5		Not used
Bit 6		Not used
Bit 7		Not used

4 1 3 Program Sequencer and Address Generator

The microengine uses ADSP-1401 for sequencing the microprogram and ADSP-1410 address generator for data memory address generation The instruction for the program sequencer comes through the instruction multiplexers (74LS257) (Figure 4 4 and 4 5) The multiplexers provides the WCS instruction when microprogram is to be downloaded (also see Section 4 1 5) During normal program execution the instruction multiplexers passes the control bits 0 to 6 from the microprogram memory to the program sequencer The address generator gets its

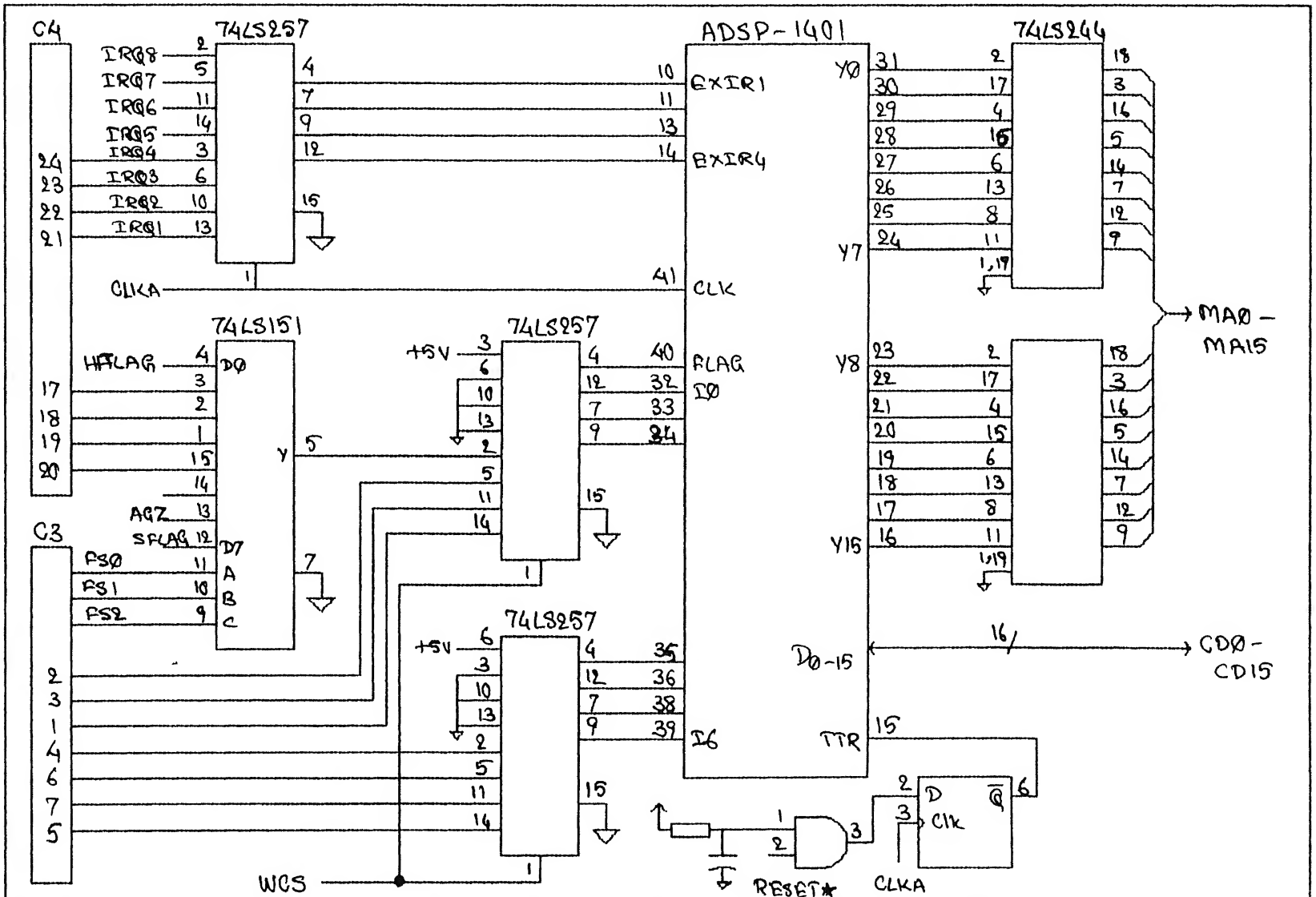


Figure 4.4 Program sequencer and associated logic

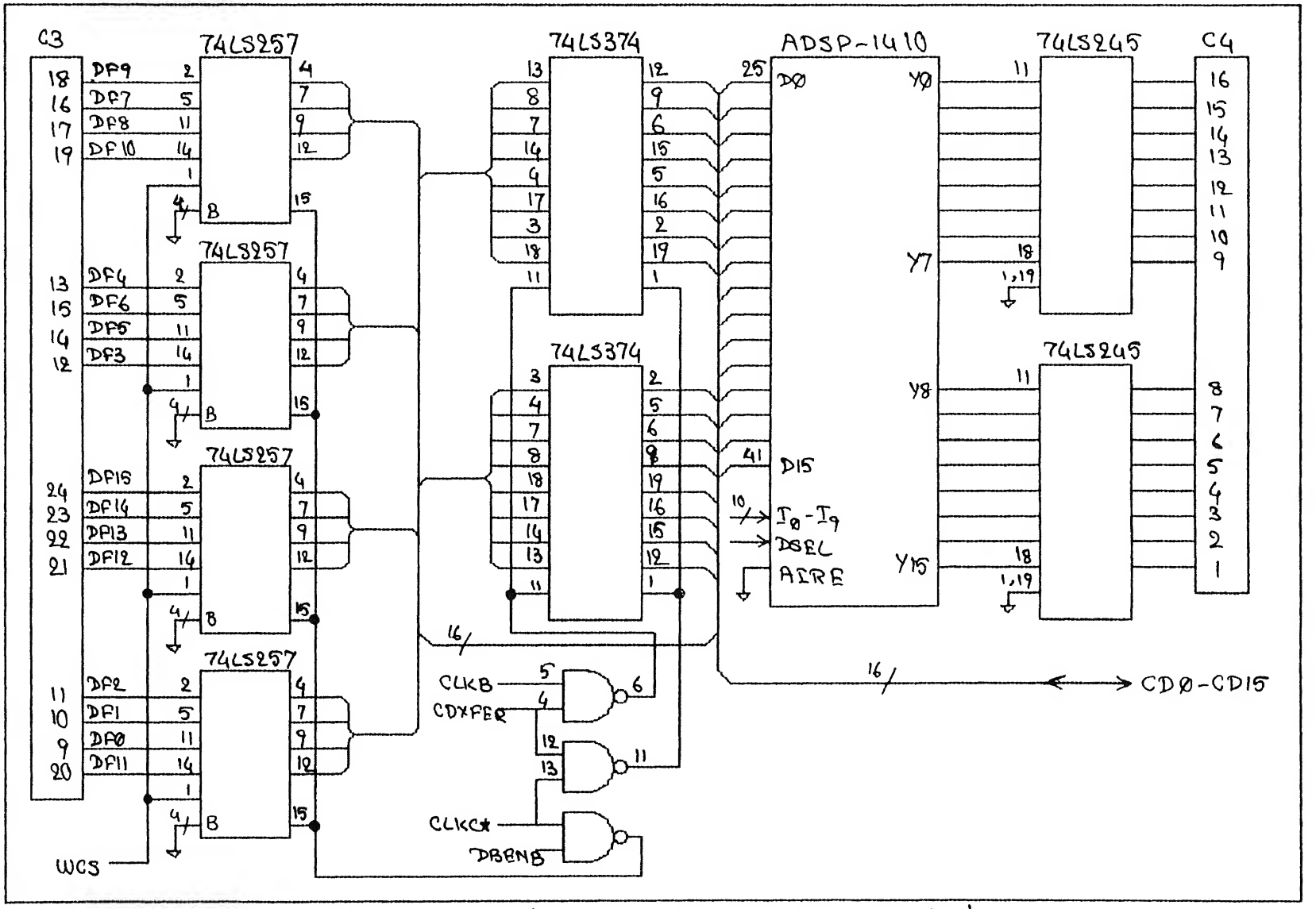


Figure 45 Address generator and cell data transfer logic.

instructions directly from the control bits 24 to 34

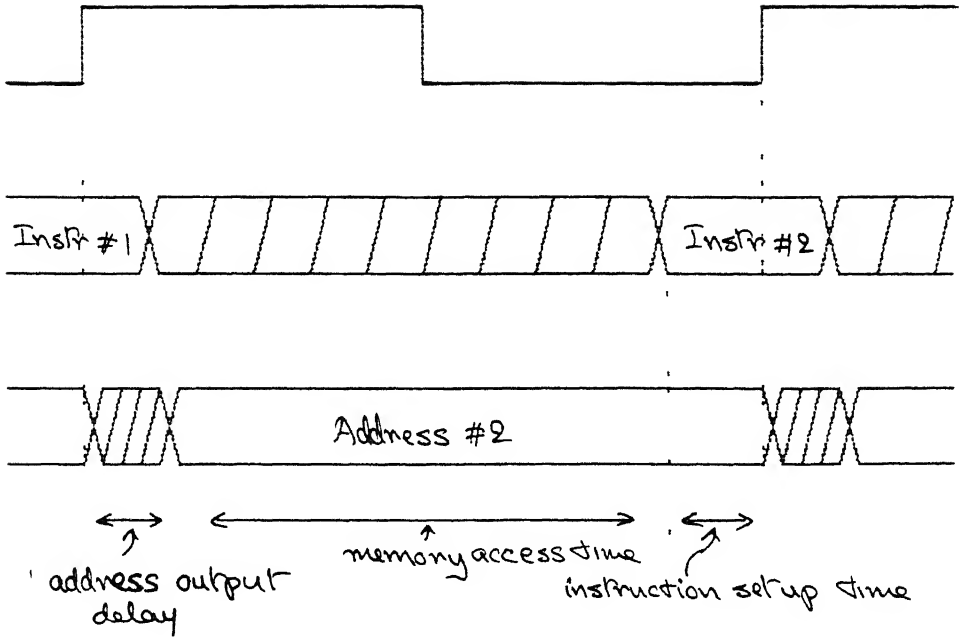
The data port of the program sequencer and the address generator are connected to the data field of the microprogram memory through multiplexers These multiplexers provides the start address (0000) during the microprogram downloading and passes the data field (control bits 8 to 23) when the sequencer is executing some microprogram The data field is 16 bit wide and permits loading of all internal registers of the program sequencer and the address generator with some desired value

The data ports of the program sequencer and the address generator act as output port during the clock high period and as input port during the clock low period Since the data ports are driven only when the clock is high there must be a provision to hold this data for the remaining clock period if we want to transfer data between the address generator and the program sequencer This is done by providing a latch (74LS374) and a control signal CDXFER (control bit no 35) so that the data is latched into the latch during the clock high period and its output is enabled during the clock low period

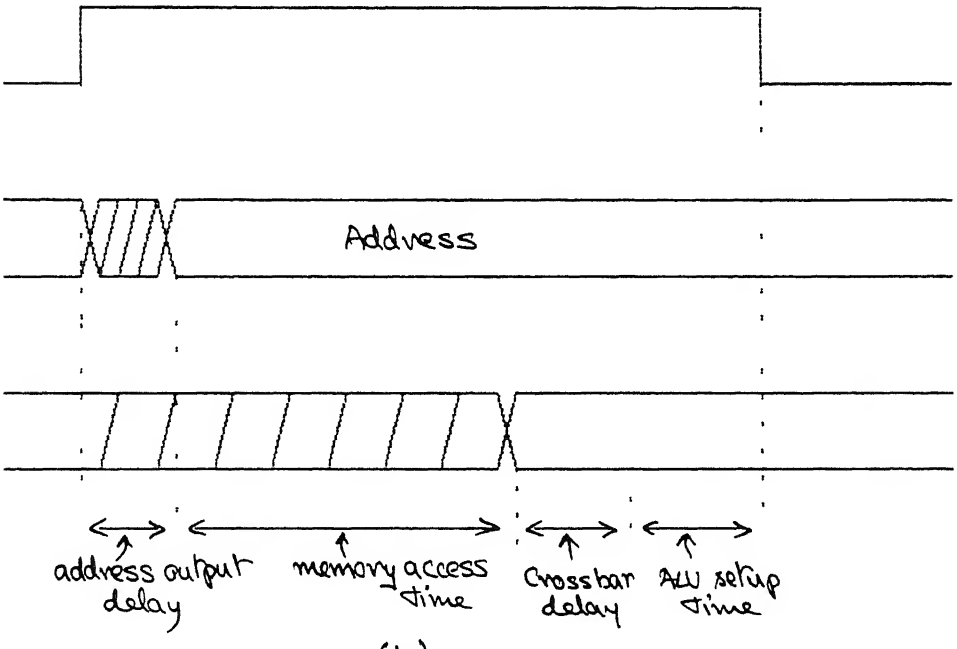
The eight interrupt lines are time multiplexed onto the four interrupt pins of the program sequencer This time multiplexing is done using a quad 2-to-1 line multiplexer 74LS257 Out of the eight interrupt lines the higher priority four lines are assigned to the host while the lower priority four interrupt lines can be used by the cell's functional units

4 1 4 Sequencer Clock Frequency

To calculate the operating frequency of the microengine one has to take a look at the sequence of events which occurs during a microprogram memory instruction fetch As shown in Figure 4 6a it involves setting up program sequencer's instruction, waiting for valid address from the sequencer, accessing



(a)



(b)

Figure 4.6 Sequencer clock frequency timings

the microprogram memory location, setting up the currently accessed instruction and so on

The worst case instruction set-up time and output delay time for ADSP-1401 are 35 nsec each The memory (HM6264LP-15) used for microprogram storage has a maximum access time of 150 nsec Therefore the smallest possible cycle time will be t_{cyc} = (35+150+35) = 220 nsec This gives the maximum operational frequency of 4 5 MHz

Unfortunately this cycle time is not sufficient for data transfer from the data memory to the ALU/multiplier registers, that loads the data on the falling edge of the clock The sequence of events for this is as shown in Figure 4 6b The address generator's clock-to-output delay has a typical value of 40 nsec The data memory has a maximum access time of 150 nsec The data set-up time for the ALU/multiplier has a minimum value of 20 nsec The crossbar introduces an additional delay of 20 nsec Taking all this into account we get $\frac{1}{2}t_{cyc}$ = (40+150+20+20) = 230 nsec Therefore the cycle time turns out to be 460 nsec In the present design we are using a 4 192 MHz crystal which is divided by two before being fed to the sequncer This frequency gives a cycle time of 476 nsec

4 1 5 Microprogram Memory

The present configuration of the cell requires 96 control bits To maintain modularity the microprogram memory is distributed onto different PCBs, each PCB containing 64 control bits The present design has two such boards and are called the *microprogram memory bank*0 and *bank*1

The microprogram memory circuit is as shown in Figure 4 7 to 4 9 It consists of HM6264LP-15 (8KX8 bit SRAM), transceiver 74LS245 and the pipeline register 74LS373 and few other logic The transceiver is used to connect the

memory to the PC data bus while downloading The pipeline register's input is connected to the memory data bits The pipeline register is clocked by the sequencer clock and is used to hold the control signals for one complete clock cycle This module of one memory chip, a transceiver and a pipeline register is replicated eight times on each PCB thereby yielding a total of 64 control bits per microprogram memory bank

The microprogram memory can be accessed either by the host or by the program sequencer When accessed by the program sequencer, the sequencer is executing some microprogram Before the start of microprogram execution the WCS bit in cell control register is set to logic zero by the host All the memory chips in different banks are permanently enabled and are in read mode The address lines MA0 to MA12 provides the address to the microprogram memory and address lines MA13 to MA15 are ignored Every clock cycle the sequencer outputs an address The content of this memory location are latched in the pipeline register and is held for one complete clock cycle

In the second case the host accesses the microprogram memory to download the microprogram Before accessing the memory the host has to set the WCS bit to logic high The host data bus being 8 bit wide the downloading has to be done sequentially one memory chip after the other During microprogram downloading the address lines MA13 to MA15 are decoded using a 3-to-8 line decoder (74LS138) to provide chip selects to one of the eight memory chips at any time (see Figure 4 8 and 4 9) The decoder itself is enabled by BANKx* control signal, which is activated by the host by setting the appropriate bits in the bank select register The same chip select signals are also used to enable the transceiver thereby connecting the memory chip to the host data bus

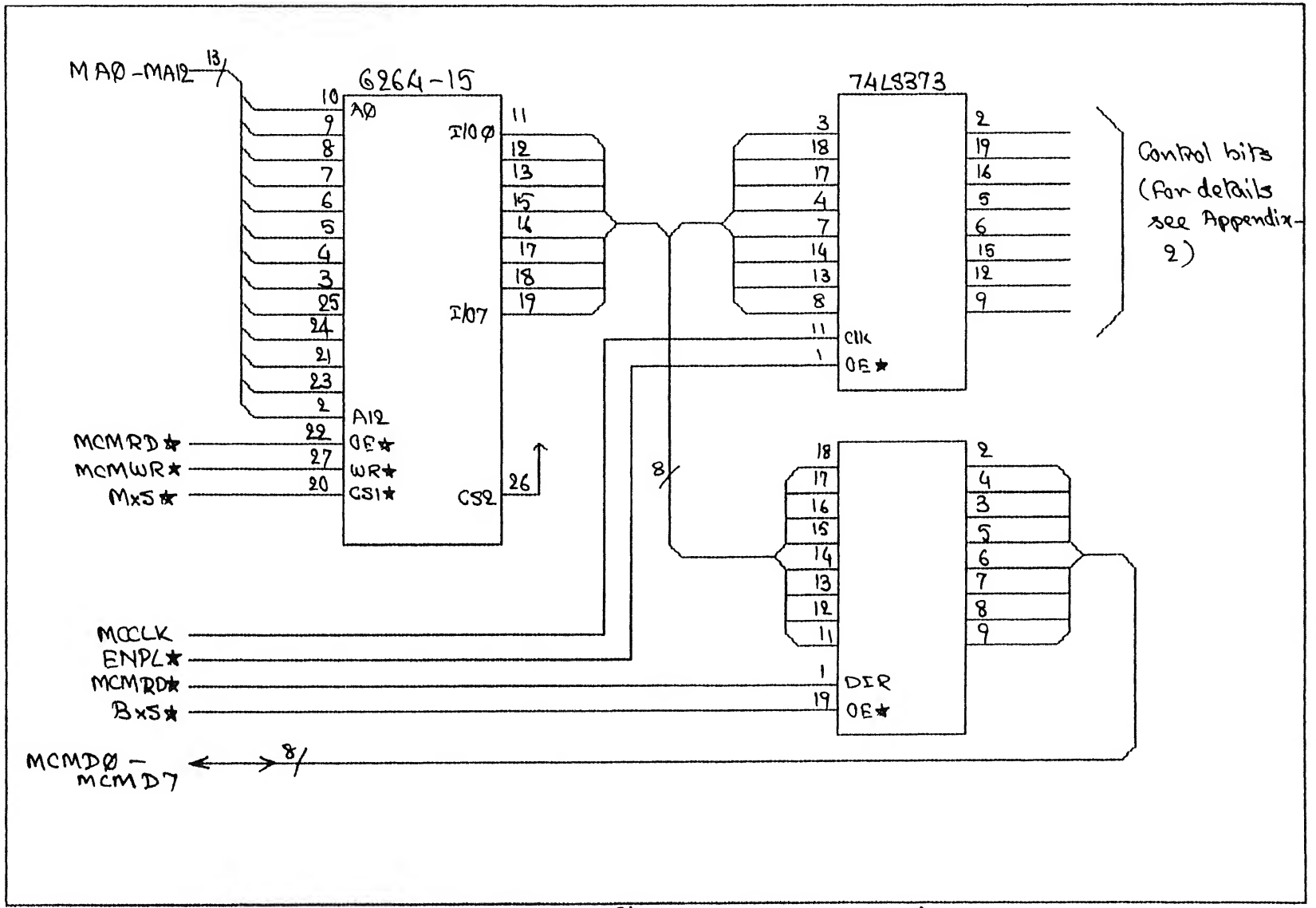


Figure 4.7 Microprogram Memory Circuit (sheet 1 of 3)

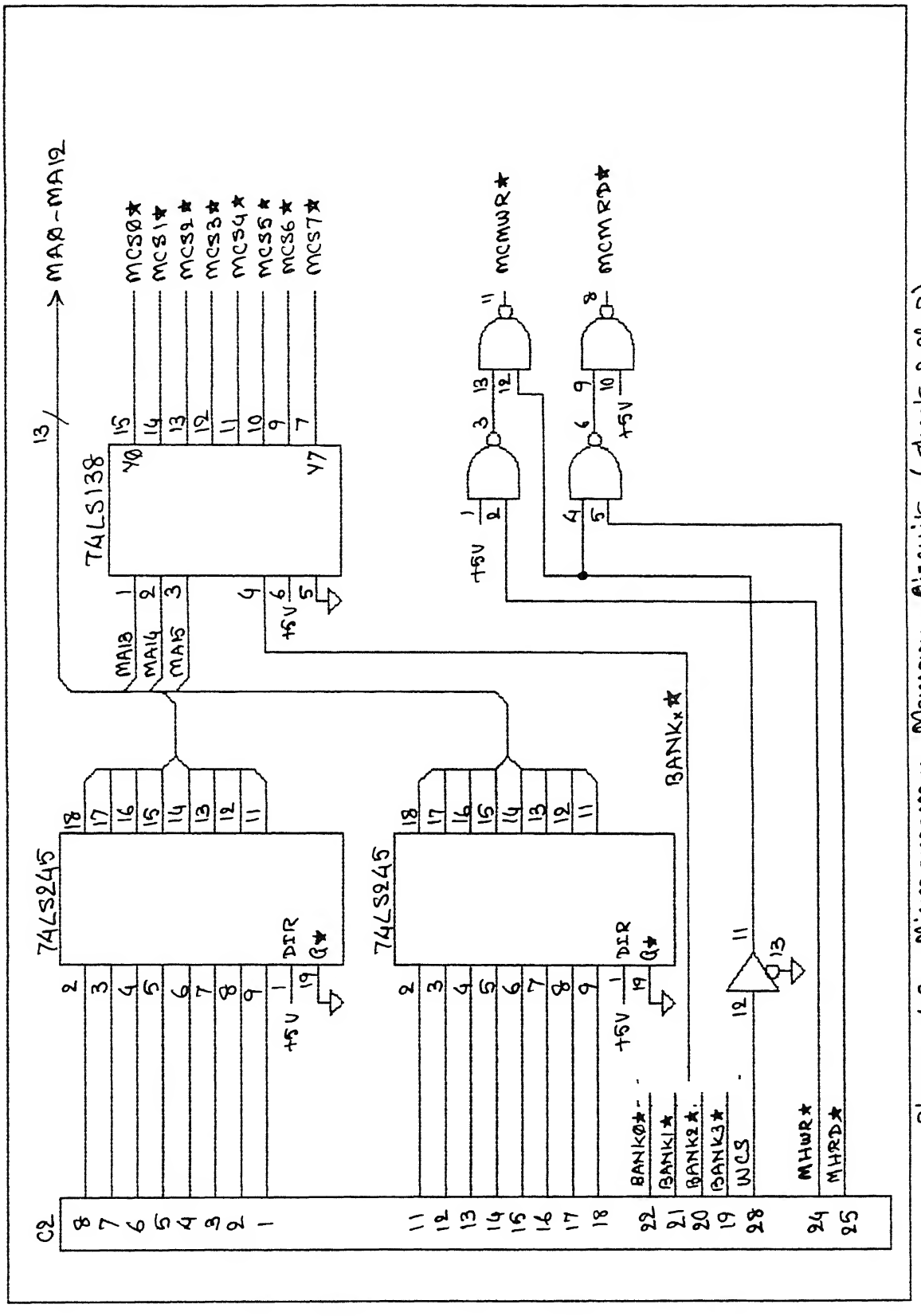


Figure 4.8 Microprogram Memory circuit (sheet 2 of 3)

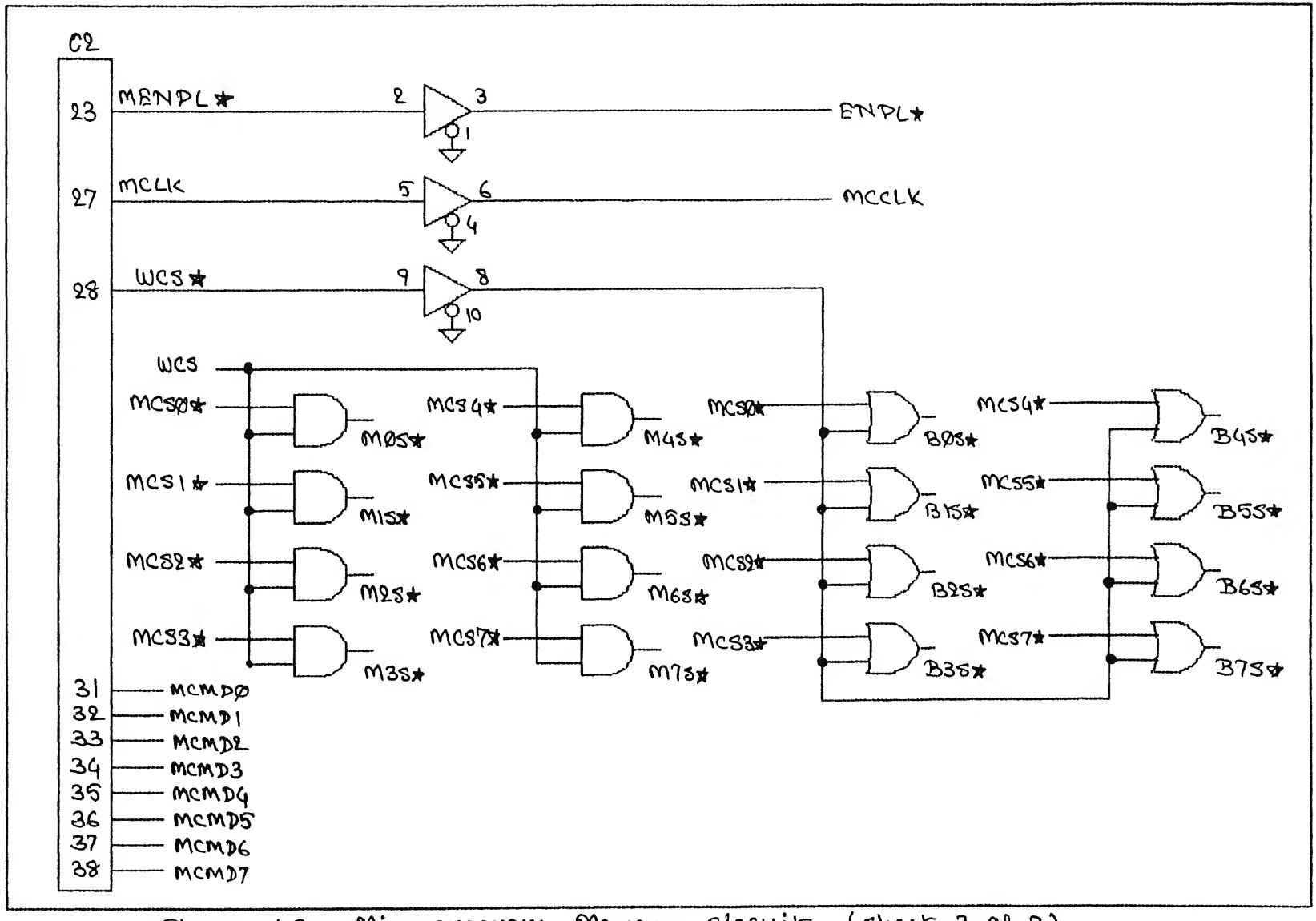


Figure 4.9 Microprogram Memory circuit (sheet 3 of 3)

4.1.6 Synchronization Circuit

Whenever the host wants to write to the microprogram memory the program sequencer provides the memory address and similarly while writing to or reading from the data memory the address generator provides the memory address. Since the host may supply data at a rate different from the program sequencer's clock rate, some handshake is required. This handshake signal should assert the FLAG input of the program sequencer after each transfer. The program sequencer will increment the address if at the rising edge of the clock it finds a logic high at its FLAG input.

The circuit which generates this synchronizing flag (SFLAG) signal is as shown in Figure 4.10. As explained in Section 4.1.1 whenever RDYEN* line goes low the RDY line on the PC bus is pulled low thereby introducing wait states. Simultaneously the IOW* ANDed with RDYEN* is synchronized with the sequencer clock using a D-type flip-flop. The synchronized host write signal (SIOW*) is fed to two D-type flip-flops in cascade which act as "one-shot". The output of this one-shot is used as host write signal (HWR*). This is done to ensure that the address lines are stable during the rising edge of the write pulse. The HWR* signal is delayed by $\frac{1}{2}$ a clock cycle to generate the SFLAG signal. The rising edge of SFLAG signal also pulls the RDY line high thereby removing the wait state in the PC machine cycle. A similar arrangement is also provided to generate the SFLAG during host read cycles. The timing diagram for the synchronizing circuit is as shown in Figure 4.11.

4.2 DATA CACHE

The cell provides for high intercell bandwidth by using two queues X and Y. To maintain a high bandwidth internal to the cell the functional units are connected through a crossbar. In order to limit the traffic through the crossbar

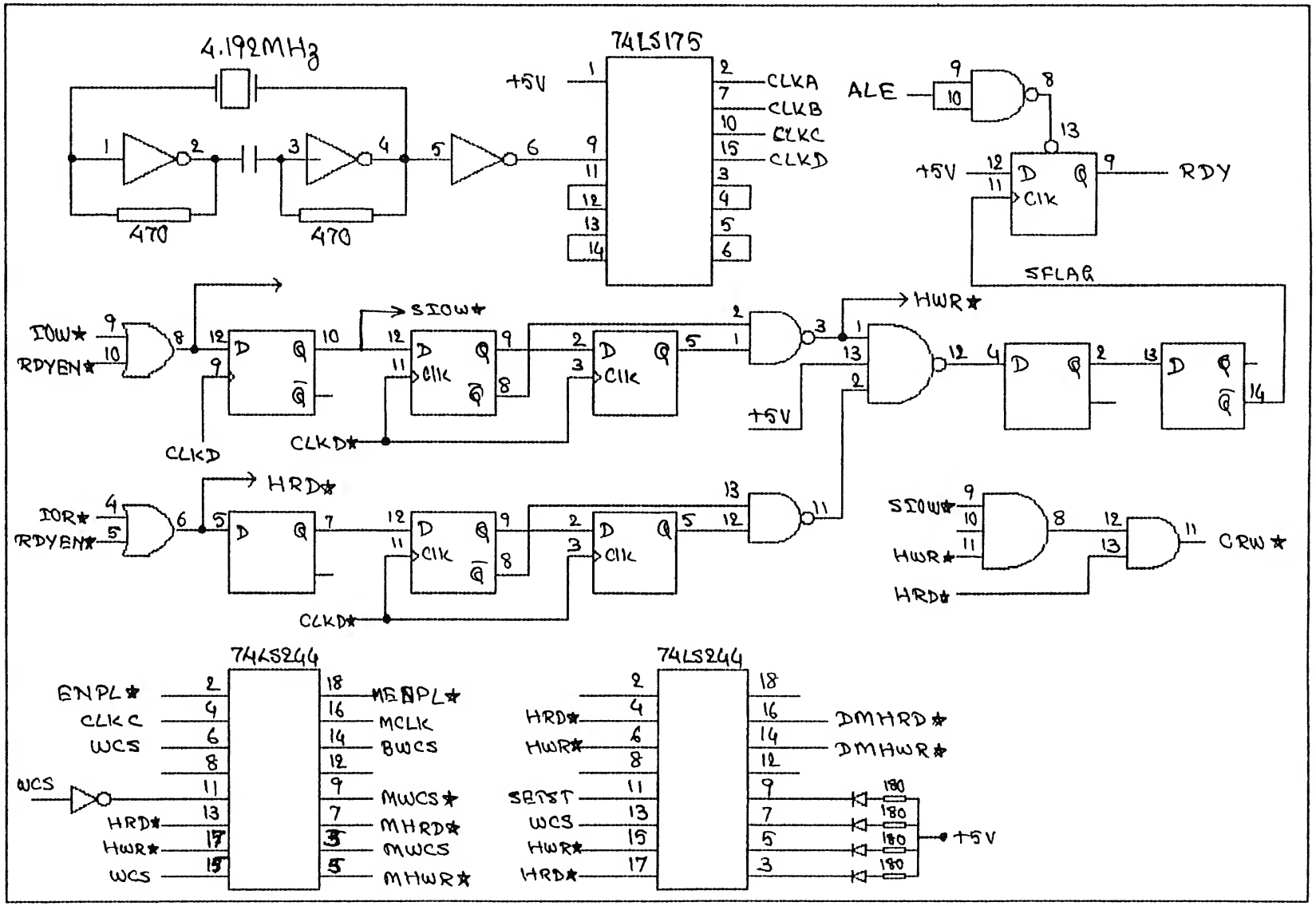


Figure 4.10 Synchronization and wait-state generator logic

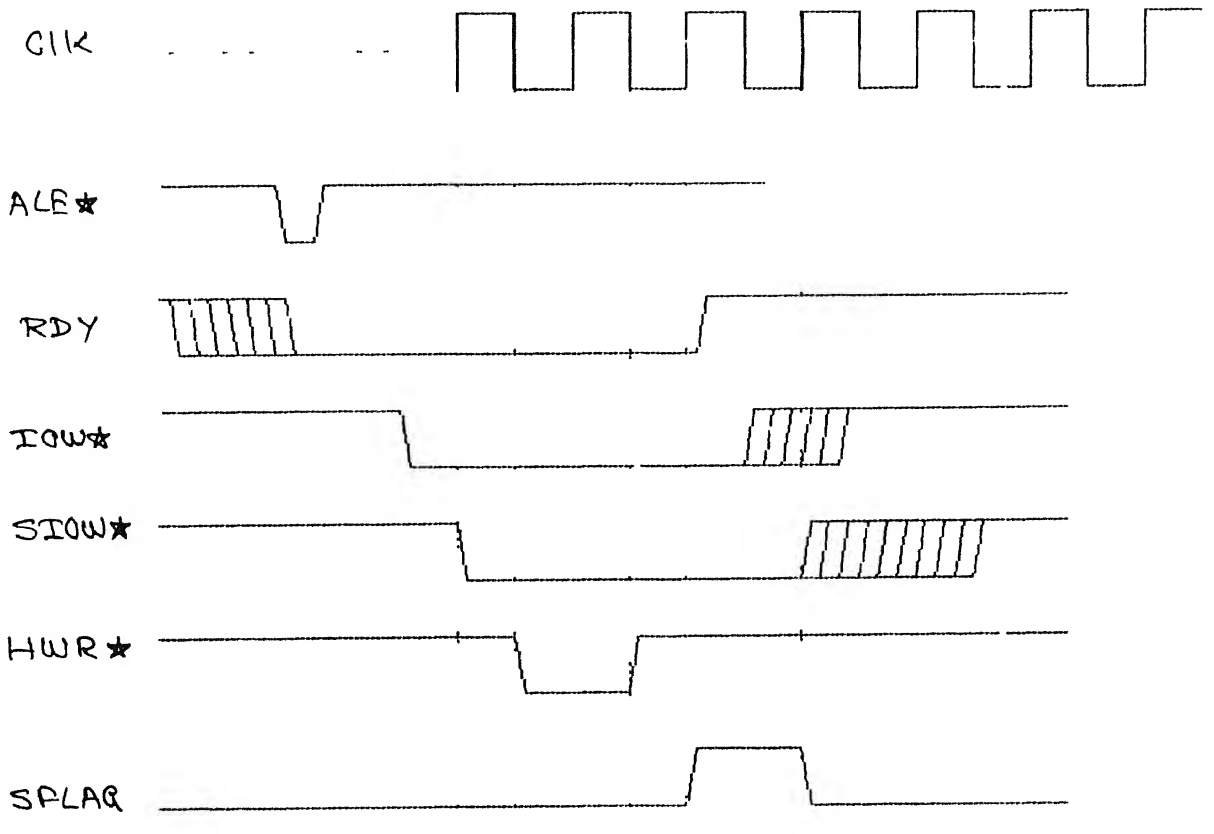


Figure 4.11 Synchronization logic timing diagram

the usage of register file and auxiliary memory was suggested in Section 3.1.6 and 3.3.1 respectively.

The register file increases the communication bandwidth by using multiport memory. The register file connection suggested in Section 3.3.2 provides - i) easy flow of data between the ALU and the multiplier, ii) delay between the transfer of the output of the ALU to the input, iii) buffering for the data to be written to the data memory.

The auxiliary memory provides a temporary storage for data with high temporal locality. This feature will be useful when a single cell has to perform the task of multiple cells. The auxiliary memory also holds frequently used constants like sine, cosine and reciprocal tables.

Due to the large number of control bits (68) required and the difficulties encountered in assembling the register file chips this module is not implemented in the current design of the cell. Presently the ALU and the multiplier are connected directly to the crossbar. This modification is explained in the next section. This section describes a possible design for the data cache module.

4.2.1 Register File

The ADSP-3128 is configurable via the DP control pin as either a 128X16 bit or a 64X32 bit register file. In the present design two such chips are "paralleled" to yield a 128X32 bit storage. Each chip has five data ports A, B, C, D and E. Data ports A and B are write-only ports, data ports C and D are read-only ports and the Edata-port is bidirectional. These five data ports allows upto six data transfer per clock cycle.

Addresses presented on the Aadr, Badr and Eadr lines during clock high are taken as the write addresses. Similarly addresses presented on the Cadr,

Dadr and Eadr lines during clock low forms the read addresses The write address latches can be made to latch the address on clock high period or transparent using the control pin Wadtrn Similarly the read address latches can be transparent or can register the read address during the clock's rising edge This can be done by making the Radtrn pin high and low respectively

The register file allows two different types of write - normal and flow-through A normal write occurs only during the clock high period The Adata-port and Bdata-port input latches can be set to transparent, latched or clock-on-falling edge mode via the ABlt and ABht control pins Similarly the Edata-port input latch can be set to transparent, latched or clock-on-falling edge mode via the Elt and Eht control pins These control pins are always registered on the rising edge of the clock and become effective as of the next falling edge This mode allows upto a maximum of three reads and three writes per clock cycle

The write flow-through mode allows a write to RAM and read from the same RAM location in clock low period If no read address matches the write address during flow-through mode, the write will not take place Write flow-through mode is enabled using the Awft, Bwft and Ewft control pins for A, B and E ports respectively During this mode of operation the write address latches should be made transparent and the read address latches can be either transparent or registered The write flow-through mode allows two flow-through writes, one normal write and upto three reads or one flow-through write, two normal writes and upto three reads per clock cycle

Each write port has an independent write inhibit control (Awinh, Bwinh and Ewinh) These control pins should be driven high whenever write operation is not desired The write ports are prioritized for both the write modes with the Edata-port having the highest priority, followed by the Adata-port followed by the Bdata-port If write to the same RAM location are attempted in a given clock

high period, the data presented at the higher priority enabled data port will be written to RAM

Read operation from register file has also two modes - normal and extra read In normal read three different locations can be read during the clock low period using the C, D and E port The data latch for these ports can be made transparent or set to register on the clock's rising edge This is done by driving the CDtran/Etran pin(s) high and low respectively Extra read from the Cdata-port, Ddata-port and or Edata-port can also be done during the clock high period Extra reads are unusual in that they occur during the clock high period which is normally a write phase Therefore each extra read takes place in lieu of one write operation to the register file This feature is enabled by making the output latches transparent i e , Rfltran and CDtran/Etran pins are driven high, for the entire clock cycle This mode allows two 16 bit words to be read from each chip per cycle through the C, D and E ports The Aadr specifies the word to be read through the Cdata-port in clock high period while the Badr does the same for Ddata-port Normal read operation allows a maximum of three reads during the clock low period In the extra read mode the register file allows four reads and two writes, five reads and one write or six reads per clock cycle Each port has also its own asynchronous three-state control Ctri, Dtri and Etri

All the control and address pins of both the register file chips are connected together The address line of all the ports will be driven by the control bits in the microprogram memory As the ALU and the multiplier put together have only three input ports extra read in a clock cycle will be of no use Therefore the control pin Rfltran is permanently grounded Further, as the Cdata-port and the Ddata-port are driving only the input port of the ALU and the multiplier they are enabled permanently by grounding the Ctri and Dtri pins The ES pin serves as a chip-select for the register file and is permanently pulled

high The circuit diagram for register file is as shown in Figure 4 12

4.2.2 Auxiliary Memory

The auxiliary memory is divided into two parts One part of the auxiliary memory is implemented with PROM (Am27PS281) and is used to hold constants like sine, cosine and reciprocal tables The other part of the auxiliary memory uses static RAM (6116) which serves as a scratch-pad memory

The RAM part of the auxiliary memory is 2048 word long and each word is 32 bit wide This RAM is addressed using a separate address generator, ADSP-1410 The RAM is enabled whenever the AMS2 control bit is high Separate read and write control signal are provided from the microcode The data port of the address generator is connected to the data field of the microprogram memory (see Figure 4 1) This allows for initializing the internal registers of the address generator The address generator gets its instruction from the microprogram memory Figure 4 12 gives the complete circuit diagram for this part of the auxiliary memory

The PROM part of the auxiliary memory is 1024 words long and holds the constant tables The first 256 locations holds the sine table, the next 256 locations holds the cosine table, the next 256 locations holds the reciprocal table and the last 256 locations are not used presently All these values are stored in single-precision floating-point format with the least 15 bits of the mantissa set to zero Unlike the RAM portion of the auxiliary memory, the PROM portion is only 16 bit wide and is split into two parts - one for the exponent and one for the mantissa The two portions of the PROM are addressed separately using latches whose inputs are connected to the outputs of the ALU and the multiplier through a multiplexer One latch is used to the latch the exponent part of the 32 bit floating-point number and addresses the exponent PROM The other

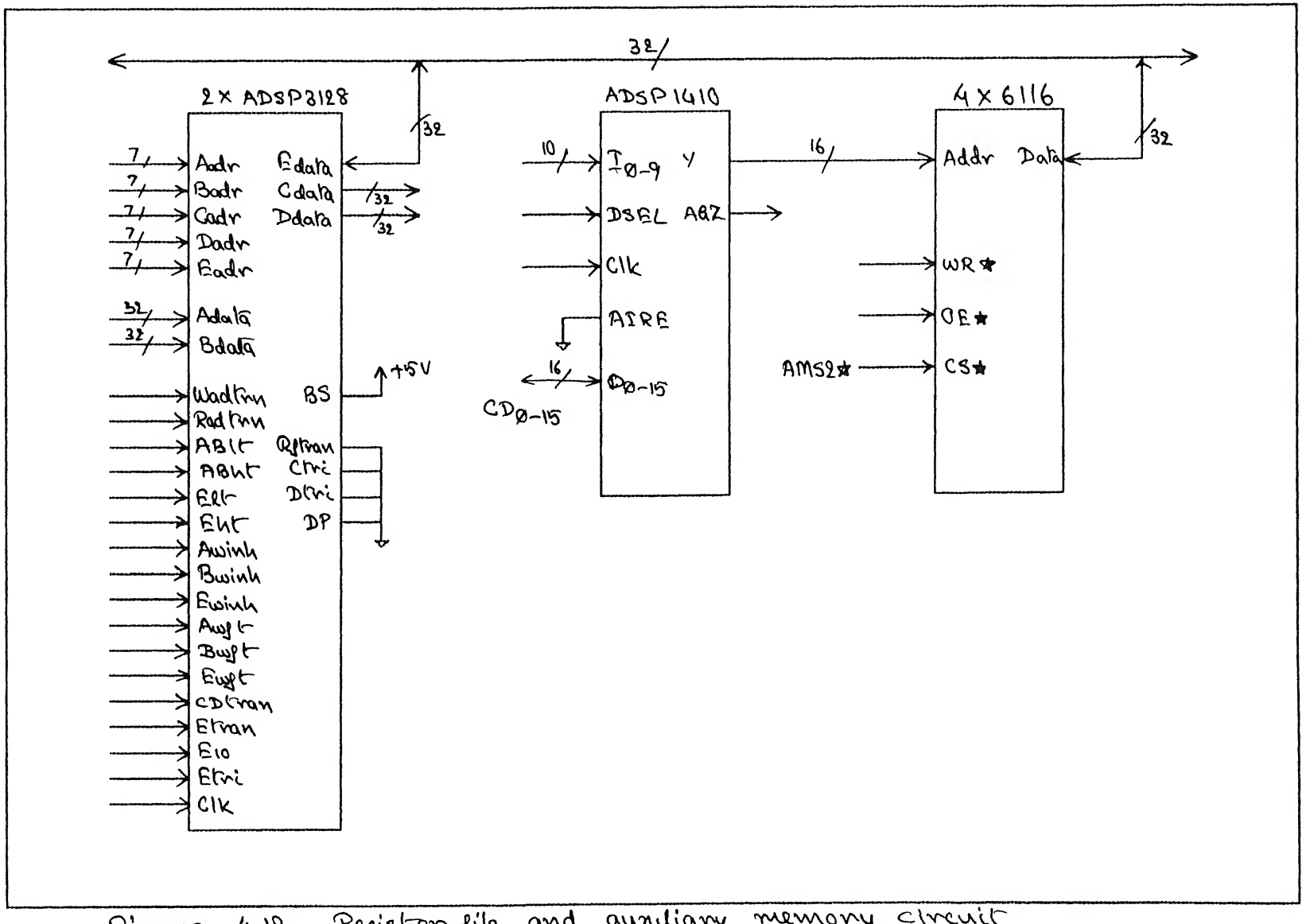


Figure 4.12 Register file and auxiliary memory circuit

latch, latches the eight most-significant bits of the mantissa and is used to address the mantissa PROM The remaining bits of the mantissa are filled with zeros The sign bit is passed on unmodified Figure 4 13 illustrates how this mapping is achieved

To select the sine, cosine and the reciprocal table from the PROM two more control bits (AMS1 and AMS0) are used The AMS1 control bit is connected to the A9 address pin of the PROM and the the AMS0 control bit to the A8 address pin The table selected depends upon these two control bits and is as listed below

AMS2	AMS1	AMS0	Usage
0	0	0	Sine table selected
0	0	1	Cosine table selected
0	1	0	Reciprocal table selected
0	1	1	Both PROM and RAM are deselected
1	0	0	RAM selected
1	0	1	RAM selected
1	1	0	RAM selected
1	1	1	RAM selected

Detailed circuit diagram of this is given in Figure 4 14

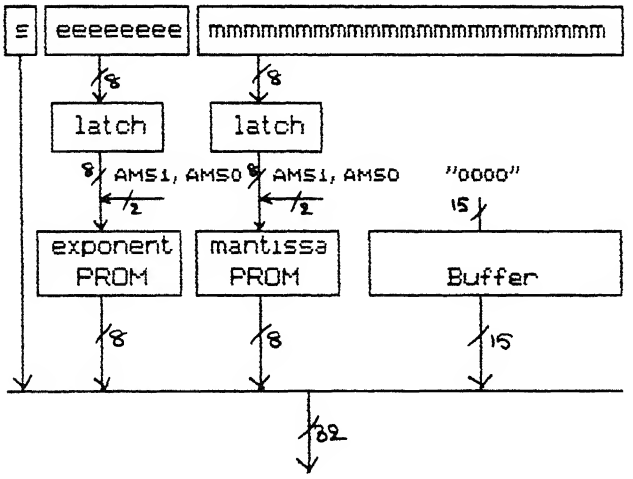


Figure 4 13 Constant table address mapping

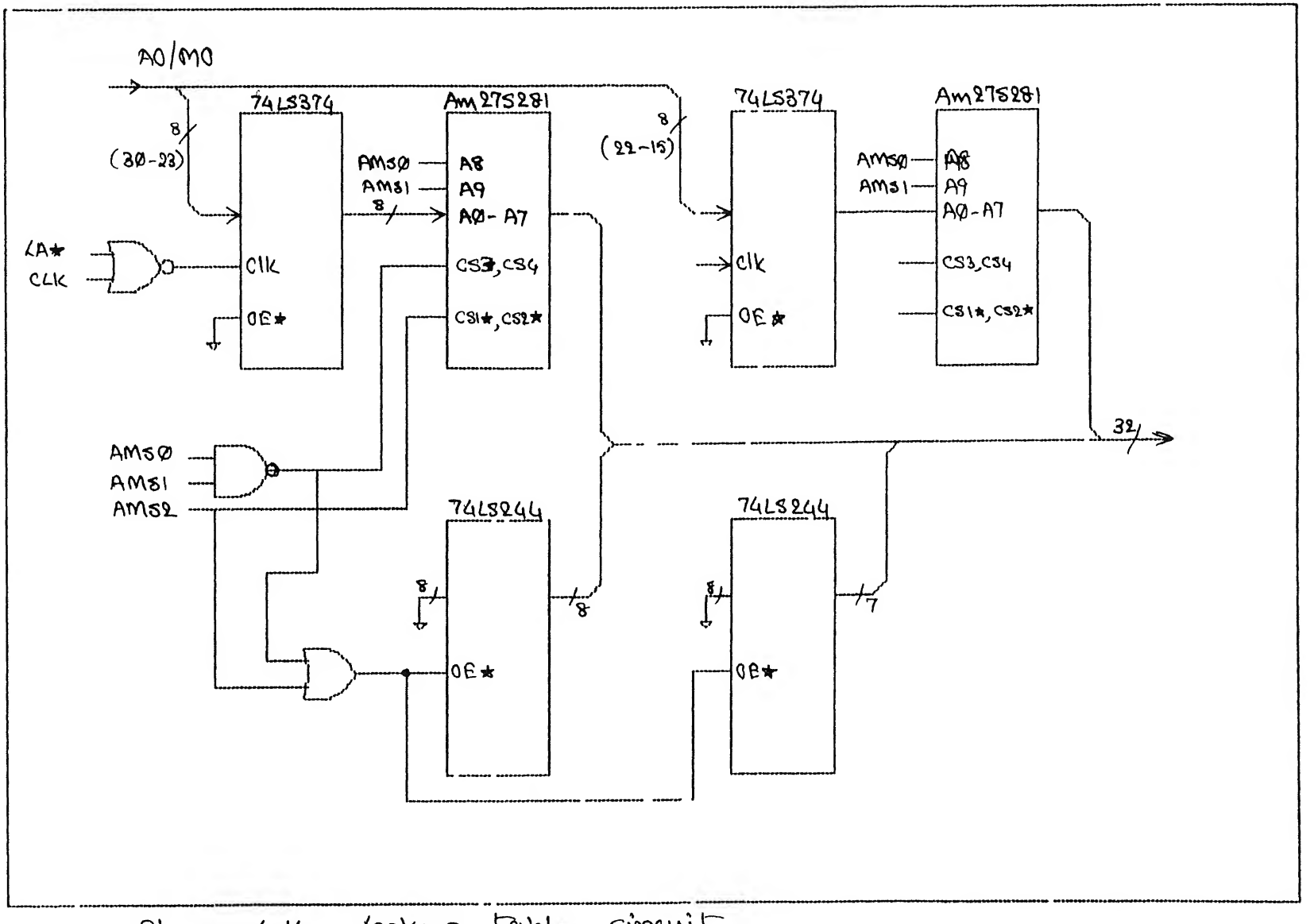


Figure 4.14 Lookup table circuit

4.3 NUMBER-CRUNCHING UNIT

The number-crunching consists of ALU, multiplier, data memory, X queue, Y queue and the crossbar circuitry. The ALU and the multipler are mounted on a PCB. The crossbar, data memory, X queue and the Y queue are assembled on two wire-wrap boards.

4.3.1 ALU and Multiplier

The ALU used in this cell is ADSP-3220. The ALU has two input ports (AIN & BIN) and one output port (DOUT), all 32 bit wide. Inside the ALU there are 8 registers, 4 of which can be loaded from AIN port and the other 4 from BIN port. 8 control pins are provided which when asserted loads the corresponding registers with the data at AIN/BIN ports. The register selection for loading the operands is done using the control bits 81 to 88. As shown in Figure 4.15 the BIN port of the ALU can be directly loaded from the crossbar output or from the output of the multiplier through a multiplexer. This input selection is done using control bit 89. The ALU has 9 bit wide instruction port which are fed from control bits 72 to 80. Along with the instruction the user has to specify which register contains the operand(s). This is done by using the RDA0, RDA1, RDB0 and RDB1 pins of the ALU. These pins are controlled by the control pins 90 to 93. Round mode control pins RND0 and RND1 are permanently connected to give round-to-nearest mode operation. For some operations the output can be 64 bit wide. The lower or upper 32 bits of the result can be output on the DOUT port by using control bit 94, which is connected to MSWSEL pin of the ALU.

The cell uses ADSP-3210 multiplier. This multiplier has a single input port (DIN) and one output port (DOUT). There are 4 registers inside the multiplier which can be loaded from the DIN port. The register(s) to be loaded is selected by asserting the SELA0, SELA1, SELB0 and SELB1 pins. These pins are connected to

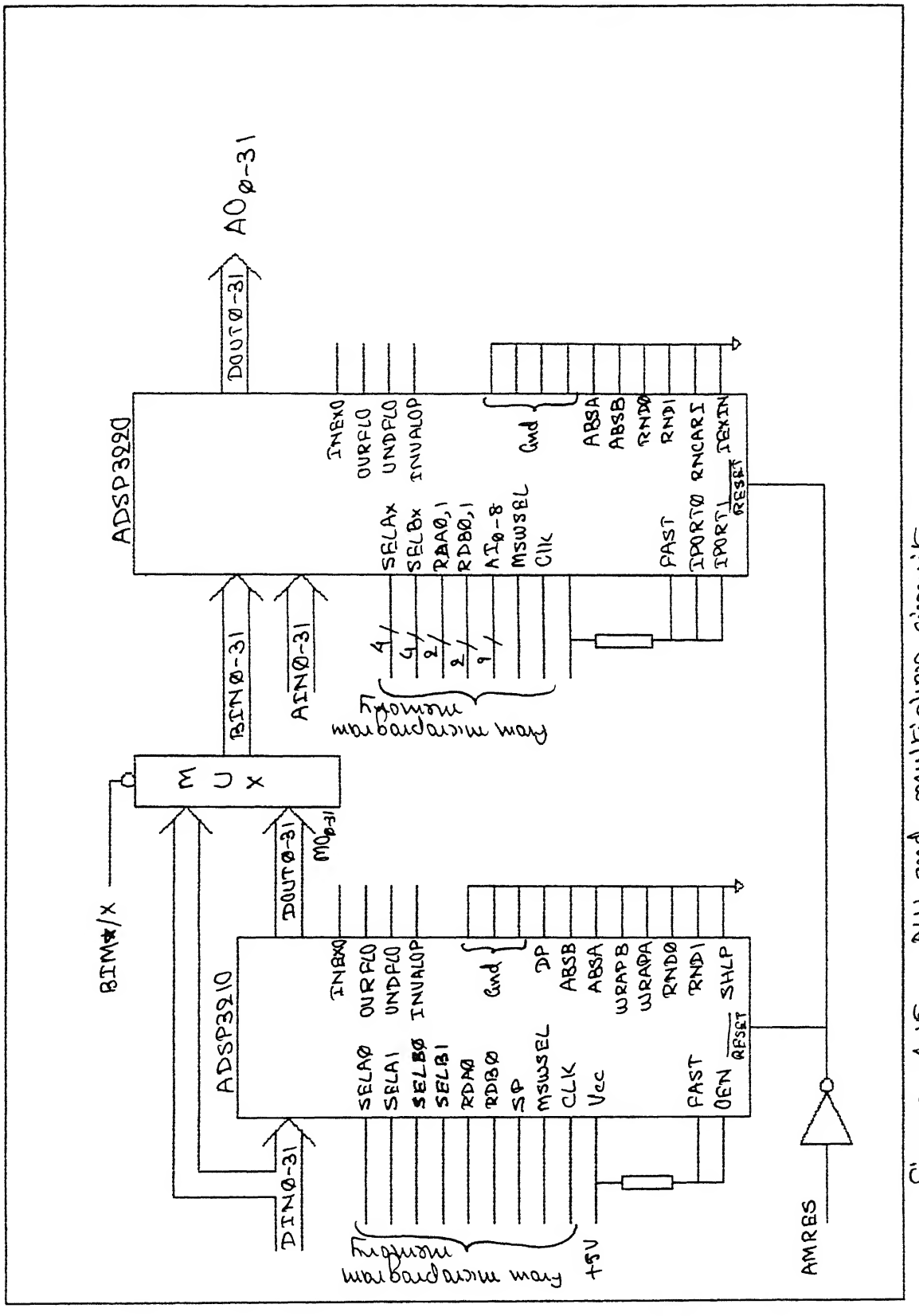


Figure 4.15 ALU and multiplier circuit

the control bits 64 to 67 Unlike the ALU the multiplier initiates one multiplication every clock cycle and has no instruction set The operand register is selected using control bits 68 and 69 which are connected to the RDA0 and RDB0 pins of the multiplier Control bit 70 connected to the SP pin of the multiplier selects the type of operand, i e two's complement or 32 bit floating-point number As in the ALU, control bit 71 selects the least significant or the most significant word of the product

4 3 2 X Queue and Y Queue

The FIFO (IDT7202A) used for X and Y queue is 1024 word long and 9 bit wide The queue width is increased to 32 bit by connecting the input control signal of four such chips The FIFO has four input control signals and three queue status signals (Figure 4 16)

The input control signals are - Read (R*), Write (W*), ReSet (RS*) and ReTransmit (RT*) The read and write signals are used for reading from and writing to the queues The RS* signal is used to reset the FIFO Asserting this signal resets both the read and write pointers to the first location The retransmit control signal, when asserted sets the internal read pointer to the first location and does not affect the write pointer This feature will be useful when the same data is to be used a number of times

The three status signals are - Full Flag (FF*), Empty Flag (EF*) and Half-full Flag (HF*) The FF* goes low whenever the write pointer is one location from the read pointer or when the read pointer is not moved and 1024 writes has taken place This flag is passed on to the adjacent cell(s) to indicate the queue status The EF* goes low inhibiting further read operations when the read pointer is one location from the write pointer, indicating that the queue is empty This signal is used by the sequencer in the cell while reading from the queue

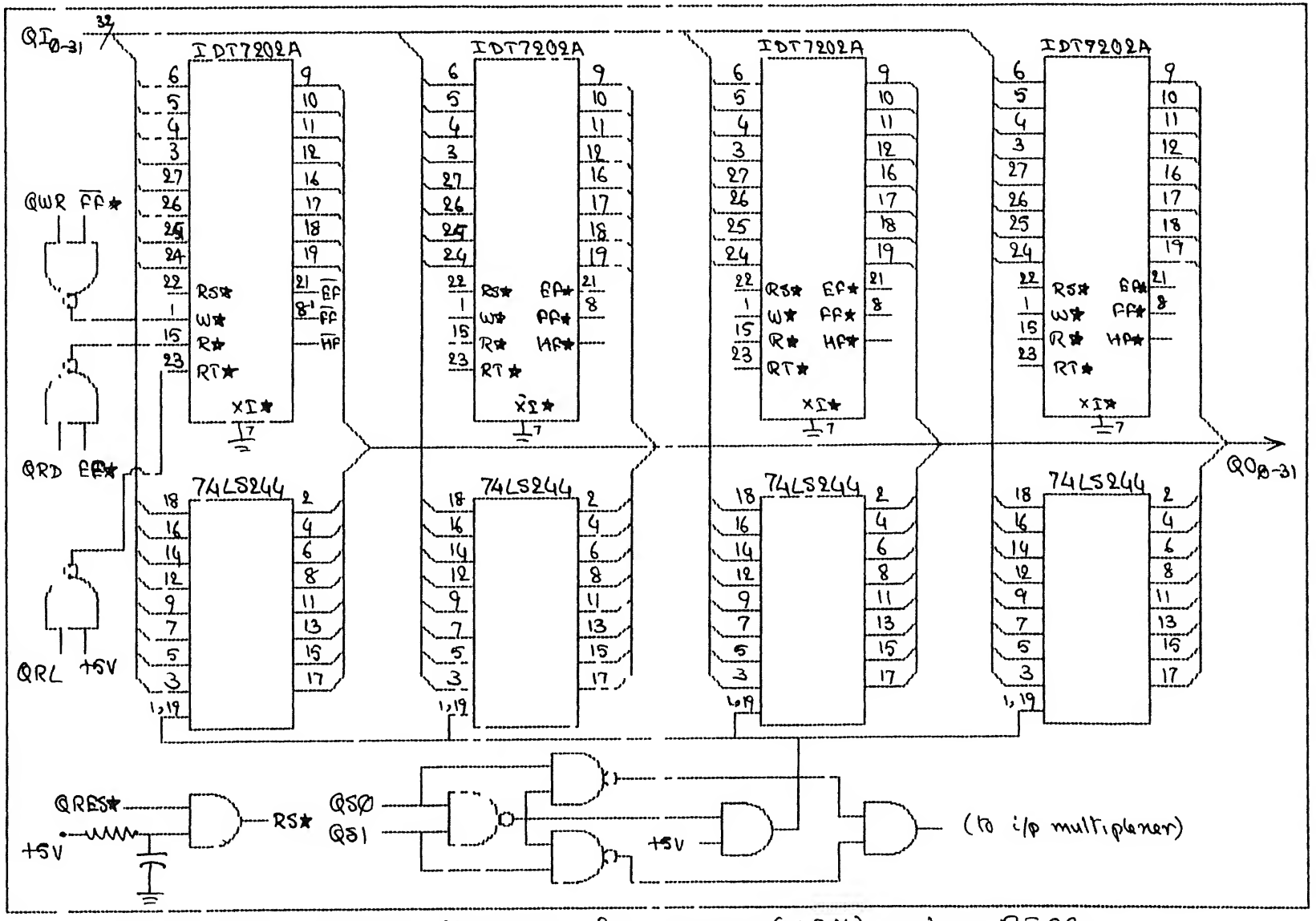


Figure 4.16 Circuit diagram for queues (X & Y) using FIFOs

The HF* output is asserted and remains so until the difference between the write pointer and the read pointer is less than or equal to one half of the FIFO size i e , 512 This signal is not used in the present design

As shown in Figure 3 2 the X queue of $cell_i$ can be written by $cell_{i-1}$ or $cell_i$ itself The multiplexer provided at the input does this multiplexing Similarly the Y queue of $cell_i$ can be written by $cell_{i-1}$ or $cell_{i+1}$ or $cell_i$ itself A 4-to-1 line multiplexer at the input of Y queue multiplexes all these inputs The output of both the queues are connected to the crossbar inputs which feeds into the ALU and the multiplier

Nonavailability of the FIFOs has forced us to use the TTL register file (74170) temporarily This register file is organized as four words of 4 bits each Eight such chips are used to form a 32 bit wide queue (see Figure 4 17)

Four data inputs are available which are used to supply the 4 bit word to be stored Location of the word is determined by the write address in conjunction with the write enable signal When the read address is made in conjunction with the read-enable signal, the word appears at the four outputs Separate on-chip decoding is provided for read and write permits This permits simultaneous read from and write to the same location

The outputs of the X queue and the Y queue are connected to the crossbar The inputs to the X queue and the Y queue are connected to the output of the ALU and the multiplier through a 2-to-1 line multiplexer (Figure 4 21) This enables loading of data into the X queue and the Y queue without using the interface unit When the interface unit and more cells are added this input multiplexer will have to be modified accordingly

4 3 3 Data Memory

The data memory circuit is as shown in Figure 4 18 and 4 19 It consists

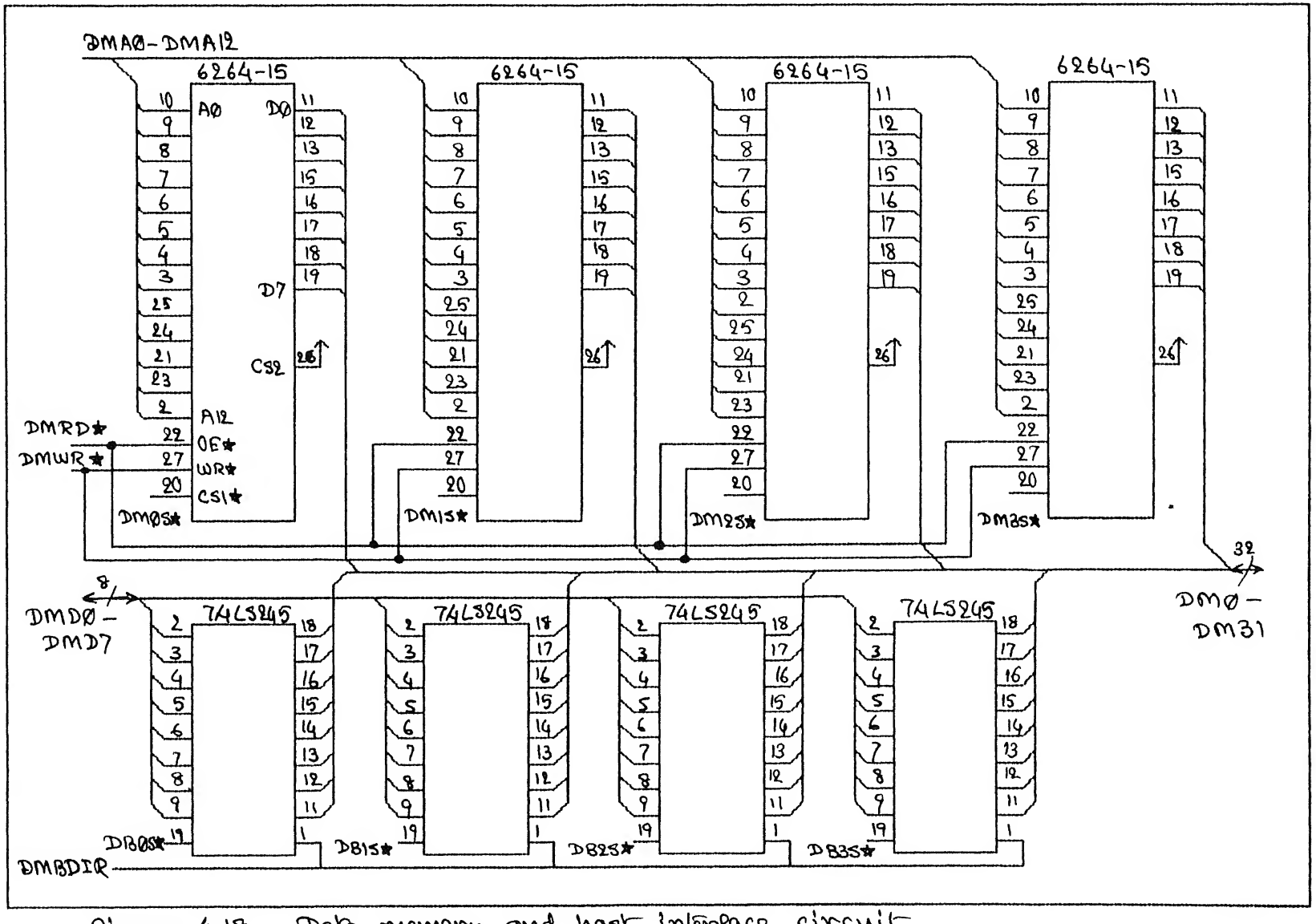


figure 4.18 Data memory and host interface circuit

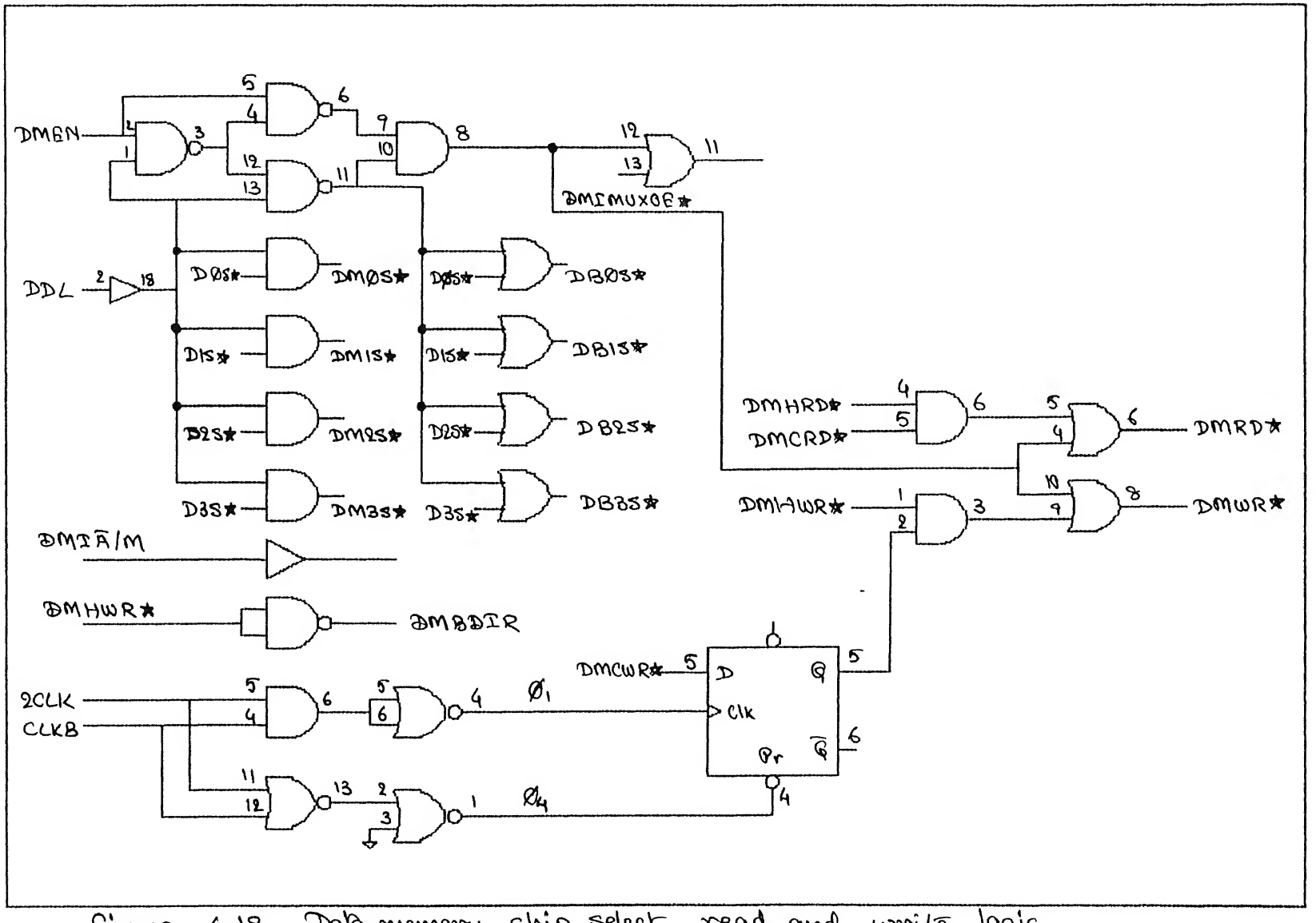


Figure 4.19 Data memory chip select, read and write logic

of four HM6264LP-15, 8K-by-8 SRAM to form a 32 bit wide data memory Also connected to the data memory are four transceivers (74LS245) which allows connection of 8 bit PC data bus to these memory chips so that the host can read from or write to the data memory The data memory can also input data from the ALU and the multiplier through a 2-to-1 line multiplexer (Figure 4 21) Also included are some logic which allows the host and the cell to access the data memory without conflict This logic is also used to multiplex the read and write signals of the host and the cell

The signals which control the sharing of the data memory by the host and the cell are DMEN (control bit 45) and DDL (control bit 44)

When the cell accesses the data memory, DMEN=1 and DDL=0, all the memory chips are enabled simultaneously during a read or write cycle During the cell access the read and write signals are obtained from control bits 42 and 43 respectively As the size of the data memory is only 8k, address lines DMA0 to DMA12 are used and the higher address bits are ignored

When the host accesses the data memory, DMEN=0 and DDL=1, the address lines DMA13 to DMA15 are decoded using a 3-to-8 line decoder to provide chip-selects to one memory chip at a time The same signal is also used to enable the transceiver, so that one memory chip is connected to the host data bus Before the host access the data memory it has to download and execute a microprogram, which will initialize the address generator The flow chart for this program is as shown in Figure 4 20 Here the synchronization between the host read/write and the sequencer clock is again achieved using the SFLAG signal as explained in Section 4 1 6 The program sequencer does not increments the program counter of the sequencer unless it senses a logic high at its FLAG input During this time the address generator simply outputs the address held in the register R0 When the flag input is asserted the program sequencer executes the next instruction in

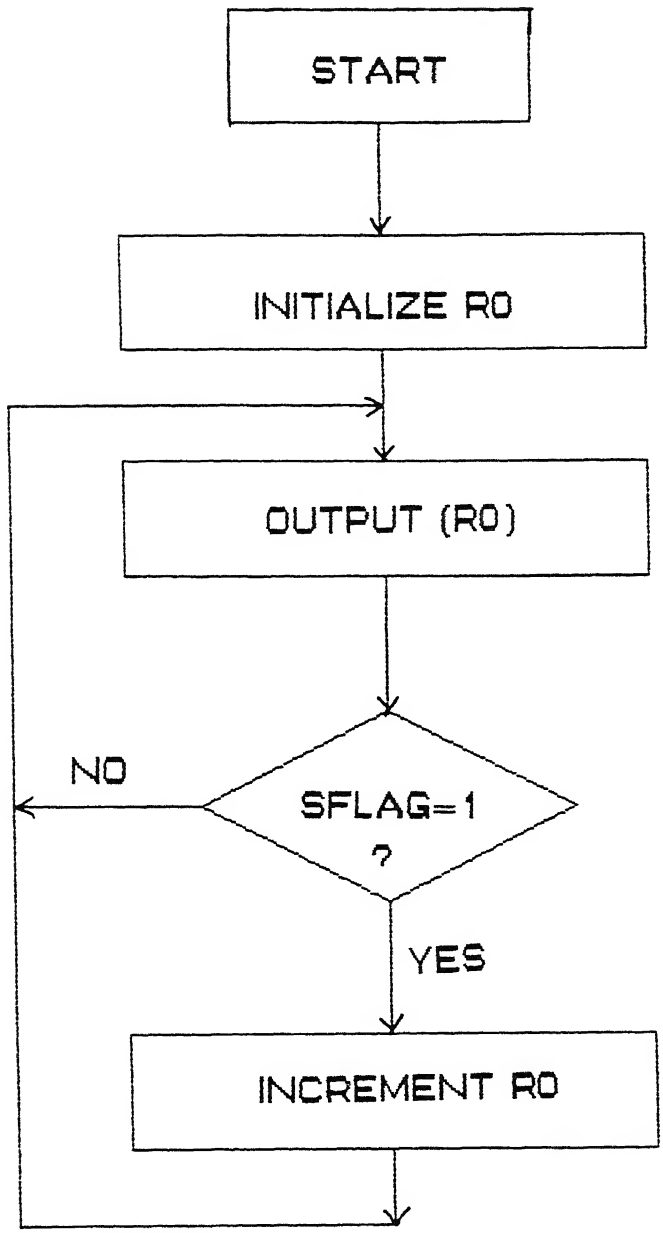


Figure 4 20 Data memory up/down load program flowchart

the next clock cycle In this clock cycle the address generator increments the contents of the R0 register by one while the sequencer loops back where it again waits for the FLAG input to be asserted

4.3.4 Crossbar

The crossbar is used to interconnect the ALU, multiplier, data memory and the queues (see Figure 4.21) The crossbar is assembled using 32 dual 4-to-1 multiplexers (Alternatively the crossbar can also be implemented with PALs or digital crosspoint switch Crosspoint switches have fixed number of ports all of which may not be needed in some application Further many such chips will have to be "paralleled" to obtain the required bus width On the other hand PALs can be programmed for the required interconnection) From hardware point of view the crossbar can be separated into two different units called the xbar1 and xbar2 Each of these crossbars has four input ports and one output port The outputs of X queue, Y queue and the data memory are connected to the three different input ports of both the crossbars

The output port of xbar1 is connected directly to the AIN input port of the ALU The output port of xbar2 is connected to the DIN input port of the multiplier and also to the BIN input port of the ALU through a multiplexer

As the fourth input port in both the crossbar is not needed by the queues and the data memory, it is utilized to provide some constant literal inputs An analysis of the distribution of constants [5] indicates that 0 and 1 are the most frequently used ones The fourth input of xbar1 holds the value 0 and that of xbar2 has the value 1

Routing of the inputs to the output is done by two control signals for each of the two crossbars Control bits 58 and 59 select the output of xbar1 and similarly control bits 60 and 61 select the output of xbar2

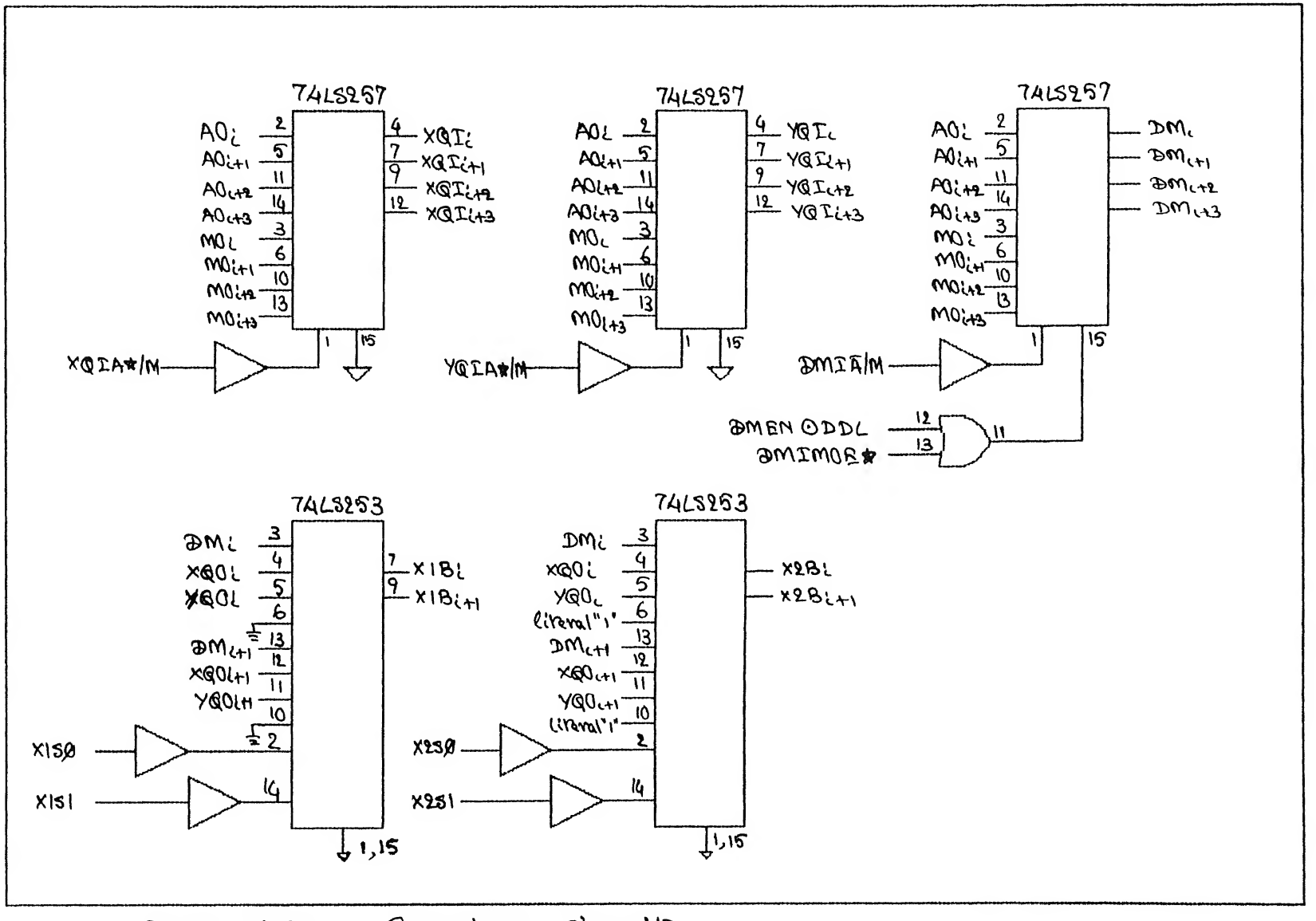


Figure 4.21 Crossbar circuit.

5. CELL PROGRAMMING

Hand coding of a microcode based system with a large number of control bits is a taxing job Appropriate software tools like meta-assembler is a must before a user can program such a system This chapter describes the instruction set of the SASF cell, followed by the software utilities developed for using the cell Finally this chapter concludes with an example of matrix-matrix multiplication program for the cell

5.1 CELL INSTRUCTION SET

As in any microcoded system, an instruction word of the SASF cell is split into multiple fields Each field controls different functional blocks in the cell To reduce the cycle time the microcode is organized horizontally without any further decoding of the control bits This approach has resulted in a wide microcode memory (96 bits) and a large set of opcodes Not all of these opcodes are valid instructions

The current implementation has one 16-bit wide data field and 23 instruction fields These instruction fields define 220 basic instructions Appendix 3 contains a list of all the valid instruction's mnemonic, opcode and it's description in brief The listing starts with the field name and the default instruction for this field which is then followed by all the instructions for this field

5.2 SOFTWARE UTILITIES

A few utility programs have been developed which are used to program

and debug the hardware A brief description of these utility programs is given below along with the program name

MEASM.EXE This program, a meta-assembler developed by Sanjay [22], is used to assemble the microprograms This assembler is a two-pass assembler In the first pass it processes a file containing the definition of the field format of an instruction word in the cell In the second phase it processes the user program translating the mnemonics, input symbols, etc , into a sequence of binary microinstructions Fields for which the user has not specified any instruction the meta-assembler automatically includes the corresponding default instruction The output of the meta-assembler is the microprogram in object code form

ARRMC COM The output of the meta-assembler is a string of 1's and 0's The length of this string equals the number of control bits in use Before downloading the microprogram to the cell the user has to rearrange this OBJ file in columnar fashion so that different columns of the object code resides in different memories in the microprogram memory bank This rearrangement of the object code is done by this program The inputs to this program is the OBJ file and the number of microcode memory chips This program then creates the output files MCBx DAT where x equals 0, 1, etc MCB0 DAT contains the microcode which is to be downloaded to BANK0, MCB1 DAT contains the microcode which is to be downloaded to BANK1 and so on

ARRD COM The data memory of the cell is 32 bit wide while that of the host (PC/XT) is 8 bit wide Transfer of data to the cell from the host has to be demultiplexed as described in Section 4 2 3 So before transfering the data has to be formated in a columnar manner This is done by ARRDM COM The output of this program is DM IN file which can be transferred to the cell as described in Section 4 2 3

RDDMTF COM The host while reading the data from the data memory

reads 8 bits at a time After reading the data from the four memory chips of the data memory, these data should be collated to form a 32 bit data This program first reads the entire 8192 bytes from each of the four memory chips in the data memory and then collates the first 512 data only (As the full data memory was not used by any program, only 512 data were collated This limitation can be removed by changing a parameter in the source file) The 32 bit data is sent to the output file DM OUT

CC COM This program acts as a controller for the cell Running on the host this program can be used for the following purpose -

i) To download the microprogram i e, transfer MCBx DAT files to the cell

ii) To download the data i e, transfer DM IN file to the data memory

iii) To reset/halt the sequencer

iv) To start execution of a microprogram

v) To fill different microcode test patterns in the microprogam memory (this feature was used while debugging the hardware in the microengine and the microprogram memory bank cards)

CELL.DAT This file defines the field formats, the field position in the microinstruction and valid instructions in each field for the cell currently in use This is a non-executable file and is used by MEASM EXE during the definition processing phase

5 3 MATRIX-MATRIX MULTIPLICATION AN EXAMPLE

Consider two matrices A and B of size M X N and N X K respectively Their product C (=AB) is a matrix of size M X K The program for this is given below The program starts by initializing the address generator's regisers R0 and R1 Register R0 points to the elements of B array which is stored in row-major

order in the data-memory As the IFU is not used currently, the elements of A matrix is initially stored in the data memory Before starting the computations the elements of A matrix are moved from the data memory to the X queue through the ALU This was done to create the systolic effect without using the IFU The X queue holds the elements of A matrix in column-major order The partial results are stored in the Y queue Final results are stored, both in the Y queue and the data-memory Register R1 points to the location where the final results will be stored in column-major order After completing the execution the program sequencer waits for the host to read the results from the data memory

As both the queues are four word deep the size of the A and B matrices is limited to 2 X 2 This matrix-matrix multiplication involves 8 multiplications and 4 additions and the number of clock cycles taken to complete them equals 15 Figure 5 1 shows the detailed activities for these 15 clock cycles Since the matrices are small in size almost half the clock cycles are lost in filling and draining the pipe From this figure we can see that starting from fifth clock cycle the multiplier outputs one product every cycle and similarly the ALU outputs the sum of two operands every clock cycle starting from the eigth clock cycle As another example consider the multiplication of A and B matrices of size 10 X 10 each This involves 1000 multiplications and 900 additions which can be completed in 1007 clock cycles if the same type of pipelining as shown in Figure 5 1 can be achieved Then at the present clock frequency of 2 096Mhz the computation rate equals 4 MFLOPS

```
, matrix-matrix multiplication program listing
start     cont & rst & clrcntr & amres
          disir & dti(10) & dsel & dbenb & 3 7
          dti(11) & dsel & dbenb & 3 16
          itr(10)(r0)  , initialize R0 register
          itr(11)(r1)  , initialize R1 register
          dtcr & dbenb & dsel & 3 0
,-----------------------------------------------------------------
, transfer the elements of A matrix to X queue
```

```
          litxb1 & laa0r & yrtr(r0) & dmtxb,A0 = "0 0"
          dmtxb & dmtxb2 & lab2rx & yrtr(r0),B2 = (R0)
          passa & aor[b2a0]
          passb & aor[b2a0]
          passa & aor[b2a0]
          passb & aor[b2a0] & atyq(w0)
          passa & aor[b2a0] & atxq(w0)
          passb & aor[b2a0] & atyq(w1)
          passa & aor[b2a0] & atxq(w1)
          passb & aor[b2a0] & atyq(w2)
          atxq(w2)
          atyq(w3)
          atxq(w3) & amres
;-------------------------------------------------------
; start matrix multiplication
          itr(10)(r0)
          enbcntr & lmra0b0 & xqtxb(r0) & xqtxb2 & yinc(c0)(r0)
          enbcntr & dmtxb & dmtxb2 & flpm[b0a0]
          enbcntr & lmra1b1 & xqtxb(r1) & xqtxb2 & yinc(c0)(r0) & flpm[b0a1]
          enbcntr & dmtxb & dmtxb2 & yqtxb(r0) & yqtxb1 & flpm[b1a0] &
               laa0r & lab1rm
          enbcntr & lmra0b0 & xqtxb(r2) & xqtxb2 & yqtxb(r1) & yqtxb1& noset &
               yinc(c0)(r0) & flpm[b1a1] & laa2r & lab3rm & aor[b1a0] & sadd
          enbcntr & dmtxb & dmtxb2 & yqtxb(r2) & yqtxb1 & flpm[b0a0] & noset &
               laa0r & lab1rm & aor[b3a2] & sadd
          enbcntr & lmra1b1 & xqtxb(r3) & xqtxb2 & yqtxb(r3) & yqtxb1 &
               yinc(c0)(r0) & flpm[b0a1] & laa2r & lab3rm & aor[b1a0] & sadd
          enbcntr & dmtxb & dmtxb2 & yqtxb(r0) & yqtxb1 & flpm[b1a0] &
               laa0r & lab1rm & aor[b3a2] & sadd & atyq(w0)
          enbcntr & yqtxb(r1) & yqtxb1 & flpm[b1a1] & laa2r & lab3rm &
               aor[b1a0] & sadd & atyq(w1)
          enbcntr & yqtxb(r2) & yqtxb1 & laa0r & lab1rm & aor[b3a2] &
               sadd & atyq(w2)
          enbcntr & yqtxb(r3) & yqtxb1 & laa2r & lab3rm & aor[b1a0] &
               sadd & yinc(c0)(r1) & atyq(w3)
          enbcntr & aor[b3a2] & sadd & yinc(c0)(r1) & atdm
          enbcntr & yinc(c0)(r1) & atdm
          enbcntr & yinc(c0)(r1) & atdm
          enbcntr & yinc(c0)(r1) & atdm
;-------------------------------------------------------
; wait for host to read the results
          yrtr(r0) & dbenb & 3 0 & dsel & hrw
wait_for_host
          jpcnf & flag7 & yrtr(r0) & noset & hrw
          jda(unconditional) wait_for_host & dbenb & yinc(c0)(r0) & hrw
          jda(unconditional) wfh & dbenb
```

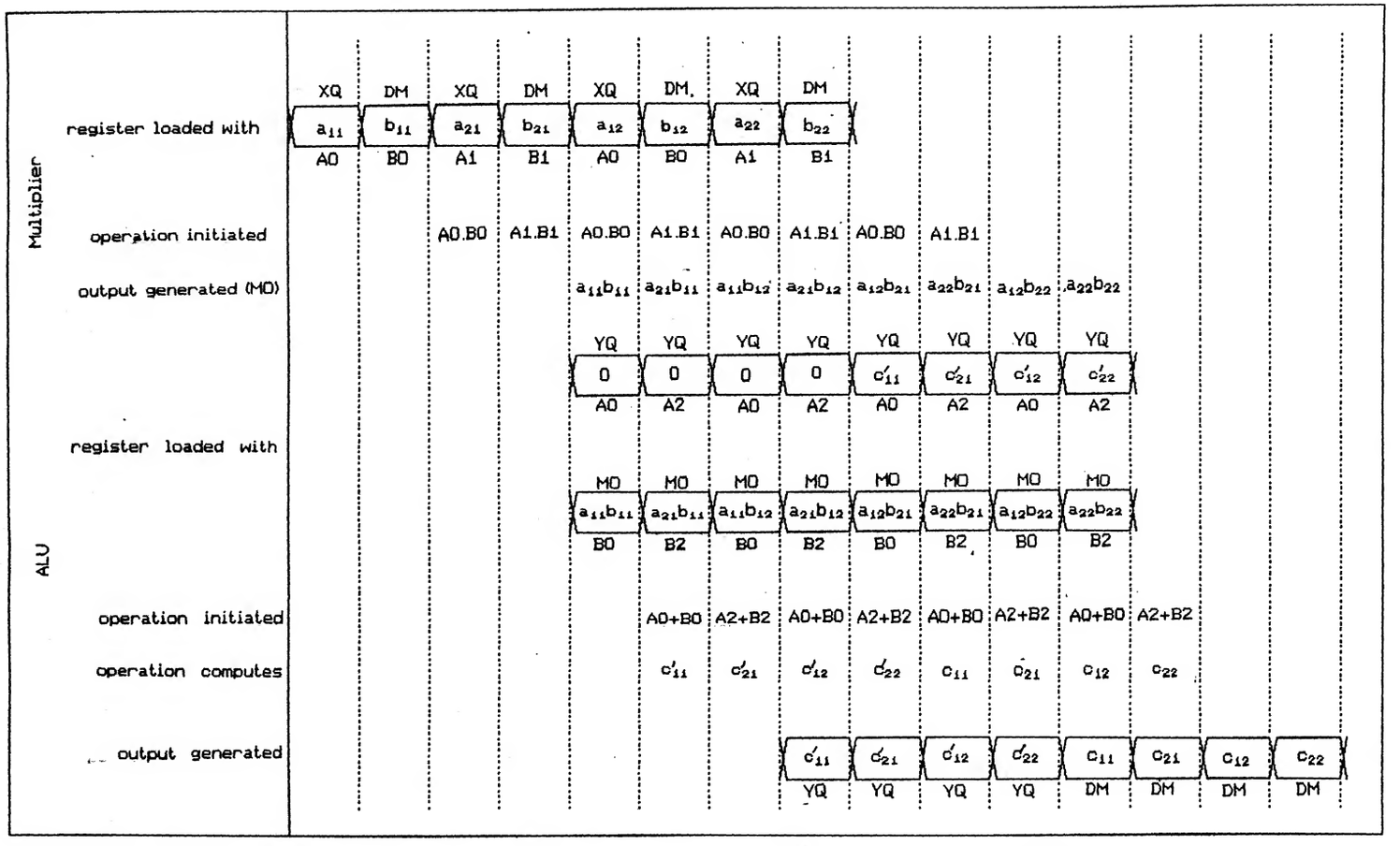


Figure 5.1 Matrix-matrix multiplication timing diagram

6. CONCLUSIONS

A systolic array is a cost-effective solution for compute-bound problems It combines the high throughput of an array processor with pipelined intercell communication Systolic arrays are ideal for fine-grain and large-grain problems Such a systolic array (SASP) was defined by Nemawarkar [19], and modified futher by Samit [21] and Usman [17] for signal processing application SASP is a linear array of microprogrammable cells connected to the external host (PC/XT) through an interface unit

In this thesis the cell architecture has been laid out in greater detail and a major part of it has been implemented using SSI MSI chips and few special-purpose ICs The cell uses 32 bit wide data paths throughout Pipelined floating-point ALU and multiplier are used to provide higher processing rate and a wide dynamic range for data Currently the cell operates at a much lower frequency, than its rated frequency of 10 MHz, which would be possible provided fast memories are used Various functional blocks of the cells were tested using different microprograms Software utilities were also developed to ease the task of programming the cell

The small size of the X and Y queue has been a limiting factor for trying out different algorithms Operating at a clock frequency of 2 096 MHz the maximum throughput of the cell can reach upto 4 MFLOPS Higher processing rate can be achieved by increasing the clock frequency and using high-speed memories

SUGGESTIONS FOR FURTHER WORK

i) A 32 bit IFU has been developed parallelly [23] The next step should be to interface the cell and the IFU With the IFU the limited size of the X and Y queues can be circumvented and the full capability of the cell can be exploited in a true systolic manner for a variety of applications

ii) More algorithms should be developed and executed on this one cell systolic array to find the limitations of the current cell architecture Based on the experience gained the cell architecture may be refined and the cell should be replicated to form a large size array

iii) Compiler which can hide the low-level details of the hardware and let the user work at a much higher level should be developed

iv) High-speed memories and FIFOs should be incorporated so that the cell can be operated at its maximum frequency

REFERENCES

[1] Allen E Charlesworth, " An Approach to Scientific Array Processing The Architectural Design of the AP-120B/FPS-164 Family ", IEEE Computer, Vol 14, No 9, 1981, pp 18-27

[2] Ashok V Kulkarni and David W L Yen, " Systolic Processing and an Implementation for Signal and Image Processing ", IEEE Tran on Computer, Vol C-31, No 10 Oct 1983, pp 1000-1009

[3] C E Leiserson, " Area-Efficient VLSI Computation ", The MIT Press, 1982

[4] Danh Le Ngoc, " Using the IDT721264/65 Floating-point Chipset ", Application Note 12, Integrated Device Technology

[5] Glenford J Myers and David L Budde, " The 80960 Microprocessor Architecture ", John Wiley & Sons, 1988, pp 25

[6] H T Kung, " Why Systolic Architectures ", IEEE Computer, Vol 15, No 1, Jan 1982, pp 37-46

[7] H T Kung and C E Leiserson, " Systolic Arrays (for VLSI) ", Sparse Matrix Proc 1978, 1979, Academic Press, Orlando, Florida, pp 256-282, also in " Algorithm for VLSI Processor Arrays," which is Section 8 3 of *Intoduction to VLSI Systems*, C Mead and L Comway, eds , 1980 Addison-Wesley, Reading, Massachusetts, pp 271-292

[8] IEEE Computer (Special issue on Systolic Arrays), Vol 20, No 7, July, 1987

[9] IEEE Task P754 Group, " A Proposed Standard for Binary Floating-point Arithmetic " , IEEE Computer, Vol 14, No 3, 1981, pp 51-62

[10] Jonathan Allen, "Computer Architectures for Signal Processing ", Proc of IEEE Vol 73, May 1985, pp 852-873

[11] José A Fortes, K S Fu, and Benjamin W Wah, " Systematic Approaches to the Design of Algorithmically Specified Systolic Arrays ", 1985 Int'l Conf Acoustics, Speech, and Signal Processing, IEEE, Piscataway, N J , pp 891-895

[12] Joseph Di Giacomo (ed), "Digital Bus Handbook ", McGraw-Hill Publishing Company, New York, 1990

[13] Juan J Novarro, et al, " Partitioning An Essential Step in Mapping Algorithms Into Systolic Array Processors ", IEEE Computer, Vol 20, No 7, July 1987, pp 77-88

[14] Kai Hwang and Fayé H Briggs, " Computer Architecture and Parallel Processing ", McGraw Hill Publishing Company, New York, 1984

[15] Marco Annaratone, et al, "The Warp Computer Architecture, Implementation and Performance ", IEEE Tran on Computer, Vol C-36, No 12, Dec 1987, pp 1523-1537

[16] Michaeal J Miller, " Understanding the IDT7201/7202 FIFO ", Application Note 01, Integrated Device Technology

[17] Mohd Usman, " Design of SASP A Systolic Array Signal Processor ", M Tech, Thesis, IIT Kanpur, April, 1989

[18] Richard J Higgins, " Digital Signal Processing in VLSI", Prentice Hall, Englewood Cliffs, NJ, 1990

[19] S S Nemawarkar, " SASP A Systolic Array Signal Processor ", M Tech Thesis, IIT Kanpur, May 1988

[20] S Y Kung, " VLSI Array Processors," Prentice Hall , Englewood Cliffs, N J , 1987

[21] Samit Choudhary " Implementation of an Interface Processor and Design of Algorithms for SASP", M Tech Thesis, IIT Kanpur, April 1989

[22] Sanjay A Wandhekar, " A Simulator for a Systolic Array Signal Processor (SASP) Based on ADSP-14XX and ADSP-32XX Chipsets ", M Tech Thesis, IIT Kanpur, Feb 1990

[23] Sqn Ldr R S Shera, " An Interface System for Digital Signal Processor ", M Tech Thesis, IIT Kanpur, April, 1990

DATA BOOKS

Bipolar/MOS Memories, Advanced Micro Devices, 1984

CMOS/BiCMOS Data Book, Cypress Semiconductor, 1989

DSP Products Data Book, Analog Devices Inc , 1988

High Performance CMOS Data Book, Integrated Device Technology, 1988

TTL Logic Data Book, Texas Instruments, 1988

APPENDIX 1

PROBLEM PARTITIONING TECHNIQUES

Clock skew puts an upper limit to the number of cells that can be incorporated in an array When the number of cells in the array is less than the PEs in the algorithm, it is necessary to carry out an additional space-time mapping The need for space-time mapping can also be validated for the reason described below

The time required to compute N components of A=BX+C, where B is an N-by-M matrix, and X and C are column vectors with dimensions M and N respectively, is of O(2N) Similarly to compute N^2 components of D=EF+G, where E, F and G are respectively M-by-N N-by-P and M-by-P matrices is of O(3N) [7] For matrix-vector multiplication, every cell is active only in alternate time intervals, and for matrix-matrix multiplication on the hexagonal array, in any row or column, out of any three consecutive PEs, only one is active at any given time If we can perform the operation of multiple cells on one PE a substantial saving in hardware can be achieved

There are two types of spatial mapping *coalescent* and *cut-and-pile* [13]

1) Coalescent mapping In this mapping, $cell_i$ for $1 \leq i \leq c$ is mapped to PE_k, where $k = \lceil i/\lceil c/L \rceil \rceil$ for $1 \leq k \leq L$ and where L is the number of PEs available In coalescent mapping (Figure A1 1) $\lceil c/L \rceil$ consecutive cells are assigned to one PE so the PE requires feedback links to itself Each data item that has entered into a PE remains in it for $\lceil c\ L \rceil$ cycles The drawback of this mapping is that the

computation load is not uniformly distributed among the PEs

2) Cut-and-pile mapping This mapping maps $cell_i$ to PE_k, where $k=1+(i-1) \bmod L$ (Figure A1 2) Each PE is functionally equivalent to a cell in the array One feedback link between the first and last PEs is required The feedback loop should have memory of size proportional to $\lceil c/L \rceil$ This mapping distributes the computing load evenly among the PEs and therefore the computing time is smaller

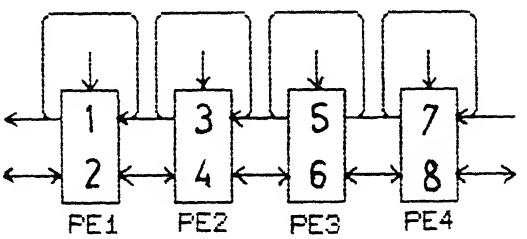


Figure A1 1 A linear systolic array with coalescent mapping

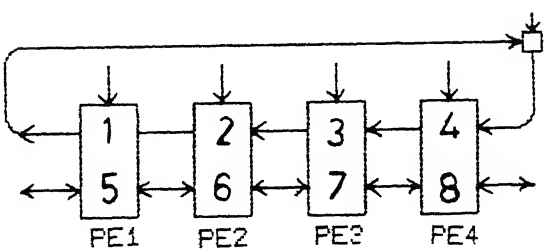


Figure A1 2 A linear systolic array with cut-and-pile mapping

APPENDIX 2

CONTROL SIGNALS AND THEIR FUNCTIONS

This appendix lists the control signals assigned to different bits of the microprogram memory word and their function in brief Also listed are the control bit numbers which will help the user in hardware debugging

Bit no	Signal name	Function
0 to 6	SI0-SI6	Program sequencer instructions
7	DBEN	Enables the output of the mux connected to the data port of the program sequencer and the address generator
8 to 23	DATA	These 16 bits holds the data which can be loaded in the internal registers of the program sequencer and the address generator
24 to 33	AG0-AG9	Address generator instructions
34	DSEL	Asserting this signal causes the data set up on the data port of the address generator to be transferred to the register specified in the instruction
35	CDXFER	Enables the latch which facilitates the transfer of data between the sequencer and the address generator
36 to 38	FS0-FS2	Selects one of the eight flag inputs to be passed onto the FLAG input of the sequencer
39	SETST	Set cell status (This signal should be used as a chip-select for the cell status register, but currently is being used to drive a LED)
40	DMIMUXOE*	Enable the output of the multiplexer, which passes on the output of ALU/multiplier to the data memory
41	DMIA*/M	Select ALU or multiplier output to be written into the data memory
42	DMCWR	Data memory write signal from cell
43	DMCRD	Data memory read signal from cell
44	DDL	Set data memory for host download/upload
45	DMEN	Enable all data memory simultaneously for cell access
46	XQIA*/M	Select ALU or multiplier output to be written into the X queue
47 to 48	XQWA,XQWB	X queue write address
49	XQWR*	X queue write signal from cell
50 to 51	XQRA,XQRB	X queue read address
52	YQIA*/M	Select ALU or multiplier output to be written into the Y queue

53 to 54	YQWA,YQWB	Y queue write address
55	YQWR#	Y queue write signal from cell
56 to 57	YQRA,YQRB	Y queue read address
58 to 59	X1S0,X1S1	Xbar1 input select signals
60 to 61	X2S0,X2S1	Xbar2 input select signals
62	LOADCNTR	Sets the clock counter to zero
63	ENBCNTR	Enables the counting operation of the counter
64	MSELA0	Load multiplier's A0 register from DIN port
65	MSELA1	Load multiplier's A1 register from DIN port
66	MSELB0	Load multiplier's B0 register from DIN port
67	MSELB1	Load multiplier's B1 register from DIN port
68 to 69	MRDA0,MRDB0	Selects the operand registers for multiplication
70	MSP	Selects floating-point or fixed point multiplication
71	MMSWSEL	Selects the msw/lsw of the multiplier output
72 to 80	ALUI0-ALUI8	ALU instructions
81	ASELA0	Load ALU's A0 register from AIN port
82	ASELA1	Load ALU's A1 register from AIN port
83	ASELA2	Load ALU's A2 register from AIN port
84	ASELA3	Load ALU's A3 register from AIN port
85	ASELB0	Load ALU's B0 register from BIN port
86	ASELB1	Load ALU's B1 register from BIN port
87	ASELB2	Load ALU's B2 register from BIN port
88	ASELB3	Load ALU's B3 register from BIN port
89	BIM#/X	Direct xbar2 output or multiplier output to BIN port
90 to 91	ARDA0,ARDA1	Selects A operand register for ALU operations
92 to 93	ARDB0,ARDB1	Selects B operand register for ALU operations
94	AMSWSEL	Select the msw/lsw of the ALU output
95	AMRES	Reset ALU and the multiplier

APPENDIX 3

CELL INSTRUCTION SET

Field #1 Program sequencer **Default cont**

cont	0000000	Continue
jpcof	0010101	If flag jump PC
jpcnf	0110101	If not flag jump PC
jtwo	101cc01	If condition jump PC+2
jda	111cc11	If condition jump data absolute
jdr	111cc01	If condition jump data relative
jdi	111cc10	If condition jump data indirect
jdrst	100111₁	If sign of C_1 jump data, C_1<=R_1, else C_1<=C_1-1
jrc	110cc11	If condition jump R_1
jrs	110111₁	If sign of C_1 jump R_1, C_1<=C_1-1
jsa	111cc00	If condition jump subroutine, absolute
jsr	111cc01	If condition jump subroutine relative
rtn	101cc11	If condition return from subroutine
branch	100cc11	If sign of C_1 jump R_1, else, C_1<=C_1-1, if condition jump data
psdss	0011110	Push data onto SS
ppssd	0111110	Pop SS to data port
wrssp	0001110	Write SSP
rdssp	0101100	Read SSP
dssp	0000010	Decrement SSP
sgsp	0000111	Select GSP
slsp	0000110	Select LSP
rdrsp	0101111	Read RSP
wrrsp	0001100	Write RSP
pspc	0100011	Push PC to RS
psgsp	0000101	Push GSP to SS
psdrs	0011111	Push data onto RS
pprsd	0111111	Pop RS to data port
a1rsp	01010₁1	Add 1 to RSP
s1rsp	0001111	Subtract 1 from RSP
s4rsp	0111100	Subtract 4 from RSP
rdsr	0101110	Read SR
wrsr	0011100	Write SR
pssr	0100001	Push SR onto SS
ppsr	0100010	Pop SR from SS
wrcntr	0111₀11	Write C_1
clrs	0010100	Clear sign bit
sets	0110100	Set sign bit
pscntr	0001₀11	Push C_1 onto SS
ppcntr	0011₀11	Pop C_1 from SS
dccntr	0110₀11	Decrement C_1
ifcdec	101cc00	If condition decrement C_1
ccir	0010001	Clear current interrupt
cair	0000001	Clear all interrupts

rtnir	0000011	Return from interrupt
rdiv	0101101	Read interrupt vector and increment IVP
wriv	0001101	Write interrupt vector and increment IVP
irmbc	0010011	IR mask bitwise clear
irmbs	0010010	IR mask bitwise set
disir	0010110	Disable interrupts
enair	0110110	Enable interrupts
slir	0010111	Select latched interrupts
stir	0110111	Select transparent interrupts
slrivp	0011101	Write SLR(=D_{5-2} and IVP(=D_{15-12}
rel16	0100100	Select 16-bit relative addressing
rel12	0100111	Select 12-bit relative addressing
rel8	0100110	Select 8-bit relative addressing
idle	0010000	Idle
ihc	0100101	Enable instruction hold control
wcs	0100000	Write control store

Field #2 Address generator **Default nop**

nop	0000000000	no operation
yinc	1011ccrrrr	Output and increment/initialize
ydec	1010ccrrrr	Output and decrement/initialize
yadd	11ccbb1rrr	Output and add offset/initialize
ysub	11ccbb0rrr	Output and subtract offset/initialize
yrtr	000101rrrr	Output and transfer R to R
yrtb	0011bbrrrr	Output and transfer R to B
yrtc	0010ccrrrr	Output and transfer R to C
dti	0000111111	Transfer D to I
itr	1000iirrrr	Transfer I to R
btr	0100bbrrrr	Transfer B to R
rtd	000100rrrr	Transfer R to D
ctd	00001100cc	Transfer C to D
btd	00001101bb	Transfer B to D
itd	00001110ii	Transfer I to D
yor	0111bbrrrr	Output & OR B with/to R
yand	0110bbrrrr	Output & AND B with/to R
yxor	0101bbrrrr	Output & XOR B with/to R
yasr	000111rrrr	Output & arithmetic shift right R to R
ylsl	000110rrrr	Output & logical shift left R to R
rst	0000000001	Reset control register
dtcr	0000101110	Transfer from data port to control register
crtd	0000101111	Transfer from control register to data port
seti	0000100iix	Set conditional re-initialization on CMP flag
setp	00001010pp	Set chip precision
sety	000001001x	Set Y port to transparent/latched mode
selr	000001101x	Select upper/lower address register bank
selb	000001100x	Select upper/lower base register bank
setu	000001011x	Set update mode (post/pre)
seta	000001010x	Set/clearconditional AIR execute mode
wra	0000101100	Write AIR with data bus
rda	0000101101	Read AIR at data bus
lda	0000011110	Load AIR from instruction port on next cycle
ydty	0000011111	Pass data bus to Y port
yrev	1001bbrrrr	Output address register in bit-reversed format

Field #3 Address generator input data select **Default nodsel**

nodsel	0	Disables address generator's data port
dsel	1	Load data from data port to the specified register

Field #4 Cell Data transfer **Default nocdxfer**

nocdxfer	0	Deselect cell data transfer latch
cdxfer	1	Transfer data from sequencer to address generator or vice-versa

Field #5 Flag input select **Default flag7**

flag7	111	Select SFLAG
flag0	000	Select HFLAG
flag1	001	(undefined)
flag2	010	(undefined)
flag3	011	(undefined)
flag4	100	(undefined)
flag5	101	(undefined)
flag6	110	Select address generator zero flag

Field #6 Set status **Default set**

set	1	Debug LED OFF
noset	0	Debug LED ON

Field #7 Data memory **Default dmnu**

dmnu	111111	Data-memory is not in use
hrw	011111	Sets data-memory for host read/writes
dmtxb	100111	Transfer from data-memory to crossbar i e, cell read
atdm	101000	Write ALU output to data-memory
mtdm	101010	Write multiplier output to data-memory

Field #8 X queue write **Default xqnu**

xqnu	1111	X queue is not in use (for write operation)
atxq(Wx)	0ww0	Write ALU output to X queue location Wx
mtxq(Wx)	0ww1	Write multiplier output to X queue location Wx (Wx = w0, w1, w2 or w3)

Field #9 X queue read **Default xqtxb(r0)**

xqtxb(Rx)	rr	Read from X queue location rr and transfer it to crossbar (Rx = r0, r1, r2 or r3)

Field #10 Y queue write **Default yqnu**

yqnu	1111	Y queue is not in use (for write operation)
atyq(Wx)	0ww0	Write ALU output to Y queue location Wx
mtyq(Wx)	0ww1	Write multiplier output to Y queue location Wx (Wx = w0, w1 w2 or w3)

Field #11 Y queue read **Default yqtxb(r0)**

yqtxb(rr)	rr	Read from Y queue location rr and transfer it to crossbar (rr = r0, r1, r2 or r3)

Field #12 Crossbar 1 **Default xqtxb1**

xqtxb1	01	Select X queue as the output of crossbar 1
yqtxb1	10	Select Y queue as the output of crossbar 1
dmtxb1	11	Select data-memory as the output of crossbar 1
litxb1	00	Select literal "0" as the output of crossbar 1

Field #13 Crossbar 2 **Default xqtxb2**

xqtxb2	01	Select X queue as the output of crossbar 2
yqtxb2	10	Select Y queue as the output of crossbar 2
dmtxb2	11	Select data-memory as the output of crossbar 2
litxb2	00	Select literal "1" as the output of crossbar 2

Field #14 Clock counter **Default disentr**

disentr	11	Disable clock-cycle-counter
enbentr	01	Enable clock-cycle-counter
clrentr	10	Reset clock-cycle-counter to zero

Field #15 Multiplier register loading instructions **Default lnmr**

lnmr	0000	Load no multiplier register(s)
lmra0	0001	Load multiplier register A0
lmra1	0010	Load multiplier register A1
lmrb0	0100	Load multiplier register B0
lmrb1	1000	Load multiplier register B1
lmra0b0	0101	Load multiplier register A0 and B0
lmra0b1	1001	Load multiplier register A0 and B1
lmra1b0	0110	Load multiplier register A1 and B0
lmra1b1	1010	Load multiplier register A1 and B1

Field #16 Multiplier instructions **Default. fxpm[b1a1]**

fxpm[b1a1]	000	Multiply [B1] and [A1], (A1 and B1 contains twos-complement integer)
fxpm[b1a0]	001	Multiply [B1] and [A0], (A0 and B1 contains twos-complement integer)
fxpm[b0a1]	010	Multiply [B0] and [A1], (A1 and B0 contains twos-complement integer)
fxpm[b0a0]	011	Multiply [B0] and [A0], (A0 and B0 contains twos-complement integer)
flpm[b1a1]	100	Multiply [B1] and [A1], (A1 and B1 contains 32-bit single-precision floating-point number)
flpm[b1a0]	101	Multiply [B1] and [A0], (A0 and B1 contains 32-bit single-precision floating-point number)
flpm[b0a1]	110	Multiply [B0] and [A1], (A1 and B0 contains 32-bit single-precision floating-point number)

flpm[b0a0]	111	Multiply [B0] and [A0], (A0 and B0 contains 32-bit single-precision floating-point number)

Field #17 Multiplier output **Default smmsw**

smmsw	1	Output most-significant-word of the product
smlsw	0	Output least-significant-word of the product

Field #18 ALU instructions **Default nop**

nop	000000000	No operation
iadd	001000011	Add A and B
isubb	001001011	Subtract B from A
isuba	001000111	Subtract A from B
iaddwc	001010011	Add A and B with carry
isubwbb	001011011	Subtract B from A with borrow
isubwba	001010111	Subtract A from B with borrow
inega	001000101	Negate A
inegb	001001010	Negate B
iaddas	001100011	Absolute value of A plus B
isubbas	001101011	Absolute value of A minus B
isubaas	001100111	Absolute value of B minus A
compla	000000101	Complement A
complb	000001010	Complement B
passa	000000001	Pass A unmodified
passb	000000010	Pass B unmodified
aandb	000010010	Bitwise logical AND of A and B
aorb	000100010	Bitwise logical OR of A and B
axorb	000110010	Bitwise logical XOR of A and B
clr	100000000	Clear all status flags
sadd	111000011	Add A and B
ssubb	111000111	Subtract B from A
ssuba	111001011	Subtract A from B
scomp	111001111	Compare A to B i e , A-B
saddas	011000011	Absolute value of A plus B
ssubbas	011000111	Absolute value of A minus B
ssubaas	011001011	Absolute value of B minus A
sfixa	011001101	Convert SP fl-pt A to twos-complement integer
sfixb	011001110	Convert SP fl-pt B to twos-complement integer
sfloata	011100101	Convert twos-complement integer A to SP fl-pt
sfloatb	011100110	Convert twos-complement integer B to SP fl-pt
spassa	011110001	Pass A unmodified
spassb	011110010	Pass B unmodified

Field #19 ALU Ax registers loading instructions **Default lnaar**

lnaar	0000	Deselect all A registers of the ALU
laa0r	0001	Load ALU register A0
laa1r	0010	Load ALU register A1
laa2r	0100	Load ALU register A2
laa3r	1000	Load ALU register A3

Field #20 ALU Bx registers loading instructions Default lnabr

lnabr	00000	Deselect all B registers of the ALU
lab0rx	10001	Load ALU B0 register from the o/p of crossbar 2
lab1rx	10010	Load ALU B1 register from the o/p of crossbar 2
lab2rx	10100	Load ALU B2 register from the o/p of crossbar 2
lab3rx	11000	Load ALU B3 register from the o/p of crossbar 2
lab0rm	00001	Load ALU B0 register from the o/p of multiplier
lab1rm	00010	Load ALU B1 register from the o/p of multiplier
lab2rm	00100	Load ALU B2 register from the o/p of multiplier
lab3rm	01000	Load ALU B3 register from the o/p of multiplier

Field #21 ALU operand registers Default aor[b0a0]

aor[b0a0]	1010	Use register B0 and A0 for source operands
aor[b2a2]	0000	Use register B2 and A2 for source operands
aor[b2a3]	0001	Use register B2 and A3 for source operands
aor[b2a0]	0010	Use register B2 and A0 for source operands
aor[b2a1]	0011	Use register B2 and A1 for source operands
aor[b3a2]	0100	Use register B3 and A2 for source operands
aor[b3a3]	0101	Use register B3 and A3 for source operands
aor[b3a0]	0110	Use register B3 and A0 for source operands
aor[b3a1]	0111	Use register B3 and A1 for source operands
aor[b0a2]	1000	Use register B0 and A2 for source operands
aor[b0a3]	1001	Use register B0 and A3 for source operands
aor[b0a1]	1011	Use register B0, and A1 for source operands
aor[b1a2]	1100	Use register B1 and A2 for source operands
aor[b1a3]	1101	Use register B1 and A3 for source operands
aor[b1a0]	1110	Use register B1 and A0 for source operands
aor[b1a1]	1111	Use register B1 and A1 for source operands

Field #22 ALU output Default samsw

samsw	1	Output most-significant-word of the ALU result
salsw	0	Output least-significant-word of the ALU result

Field #23 ALU/multiplier reset Default amopr

amopr	0	ALU and multiplier are operational
amres	1	Reset ALU and multiplier

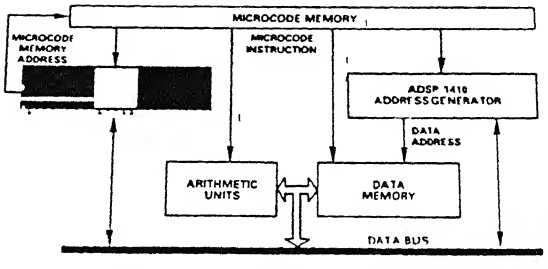
APPENDIX 4

ANALOG DEVICES

Word-Slice™ Program Sequencer

FEATURES

- **16-Bit Microcode Addressing Capability**
- **Look-Ahead™ Pipeline**
- **Extensive Interrupt Processing, With Ten On-Chip Interrupt Vectors**
- **70ns Cycle Time, 25ns Clock-to-Address Delay**
- **64-Word RAM for Storing**
  - **Subroutine Linkage**
  - **Jump Addresses**
  - **Counters**
  - **Status Register**
- **375mW Maximum Power Dissipation with CMOS Technology**
- **48-Pin DIP**



WORD SLICE™ MICROCODED SYSTEM WITH ADSP 1401

ENERAL DESCRIPTION

he ADSP-1401 is a high-speed microprogram controller op-mized for the demanding sequencing tasks found in digital gnal processors and general purpose computers In addition to gh speed (25ns clock-to-address delay) and large addressing nge (64K of program memory), this Word-Slice component s unique features that make it highly versatile

- on-chip storage and control of ten prioritized and maskable interrupts
- four decrementing event counters
- absolute, relative and indirect addressing capability
- download capability (writeable control store) and
- a dynamically configurable 64-word RAM

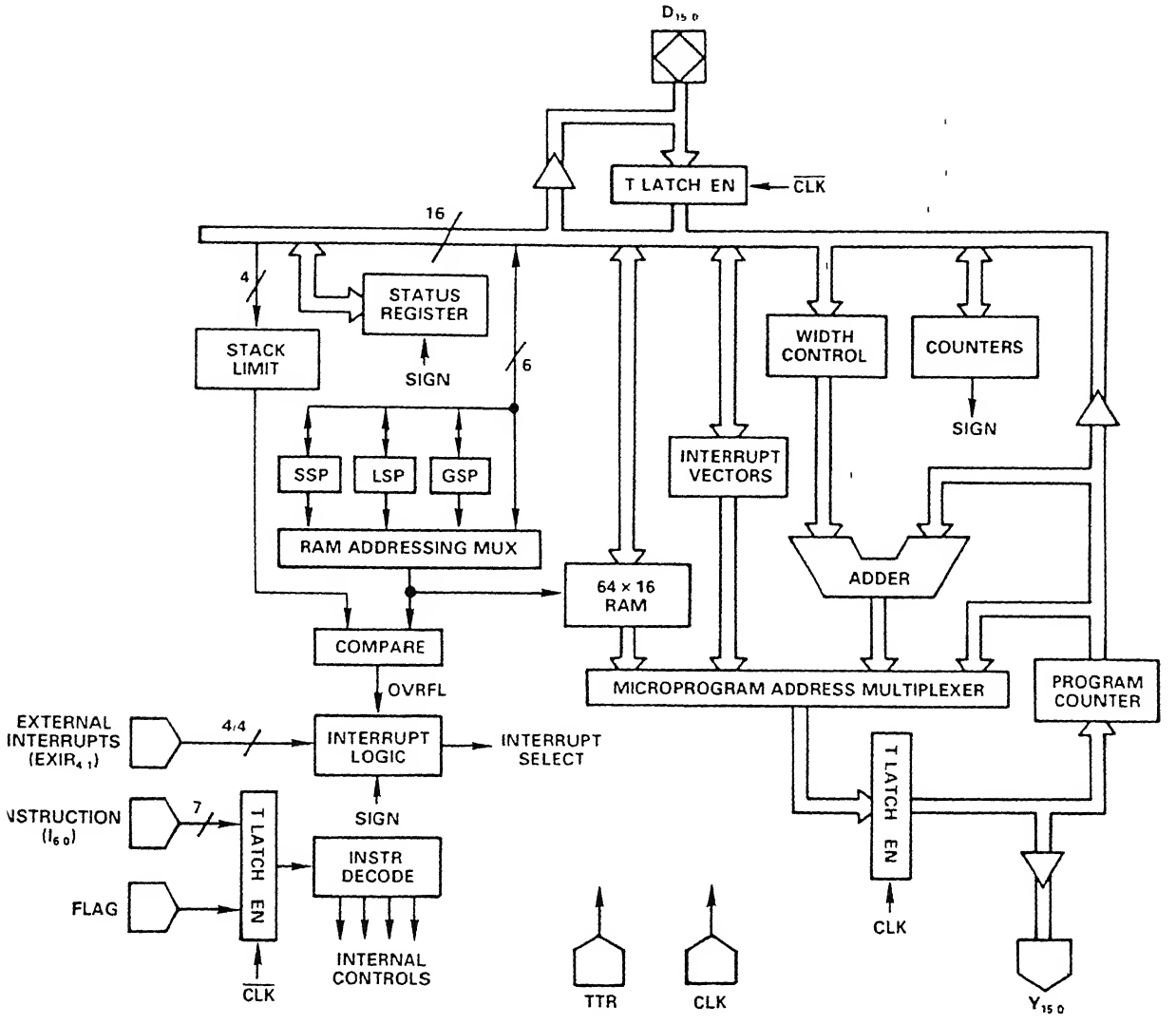
e ADSP-1401 microprogram sequencer's main task is to ovide the appropriate microprogram addressing to support gramming requirements (e g , looping, jumping, branching, routines, condition testing and interrupts) An internal Look-ead pipeline, controlled by both phases of the clock, allows ADSP-1401 to satisfy these requirements at very high speed

ring each micro-instruction, the ADSP-1401 monitors the ditions and instructions to determine the next microprogram ress This address can come from one of several sources the k, the jump address space in the RAM, the data port, the rrupt vectors, or the microprogram counter An extensive of conditional instructions are also available, including jumps, iches, subroutines, interrupts, and writeable control store

The ADSP-1401's internal 64-word RAM is user-configurable into three regions, subroutine stack, register stack and indirect jump address space The subroutine stack is used for linking interrupts and subroutines and, during their execution, allow storage of system states The register stack allows association of unique jump addresses with various levels of interrupts and subroutines (both local and global stacks are provided) Indirect jump capability is also supported, addressing for which is provided at the data port

Interrupts are handled entirely on chip The ADSP-1401's internal interrupt control logic includes registers for eight external (user) interrupt vectors, a mask register, and a priority decoder Two additional vectors are reserved for internally-generated interrupts resulting from counter underflow and stack limit violation A stack limit violation is caused by stack overflow, underflow or collision A mechanism is provided for recovering from stack violations

The ADSP-1401's four decrementing 16-bit counters are used to track loops and events These counters generate a signal when negative This negative condition is used by several conditional instructions and can also trigger an internal interrupt



ADSP 1401 Block Diagram

ADDRESSING MODES

- Direct both absolute and relative
- Indirect from internal RAM

HARDWARE FEATURES

- Instruction Port
- Bidirectional Data Port
- Four Input Address Multiplexer
- Three Stack Pointers
- Four Event Counters
- Condition Flag
- Eight Prioritized and Maskable User Interrupts
- TTR Pin
  - Trap
  - Three-State
  - Reset

INSTRUCTION TYPES

- Jumps and Branches
- Stack Operations
- Status Register Operations
- Counter Operations
- Interrupt Control
- Relative Address Width Controls
- Instruction Hold Control
- Writeable Control Store
- Dedicated Counter Underflow Interrupt
- Dedicated Stack Overflow Interrupt

ADSP-1401 PIN ASSIGNMENTS

Pin Name	Description
$I_6 - I_0$	The 7-bit microinstruction controlling the ADSP-1401
$Y_{15} - Y_0$	Output bus which provides addresses to the microprogram memory
$D_{15} - D_0$	Bidirectional Data bus for transferring data to or from the ADSP-1401
$EXIR_{4-1}$	Four external interrupt request lines Note that internal circuitry supports 8 interrupts with the aid of an external 2 to 1 multiplexer
CLK	External clock input
FLAG	An input used for conditional instructions Its source is usually a condition multiplexer
TTR	A multi-purpose pin accommodating traps, output disable and reset
V_{DD}	+ 5 Volt supply
GND	Ground

DEVICES

Address Generator

FEATURES

- **16-Bit Addresses with Higher Precision Options**
- **High Speed, Clock-to-Valid-Address Delay of 20ns**
- **Look-Ahead™ Pipeline**
- **Versatile Addressing Hardware**
  - **30 16-Bit Registers**
  - **16-Bit ALU with Left/Right Shift & Carry I/O**
  - **Comparator**
  - **Bit Reverser**
- **Dual Ports**
- **Powerful Single-Cycle Looping Instructions**
- **175mW Maximum Power Dissipation with CMOS Technology**
- **48-Pin DIP**

GENERAL INFORMATION

The ADSP-1410 is a fast, flexible address generator optimized for digital signal/array processors and other high performance computers. This low-power CMOS device rapidly generates the data memory addresses required by routines such as digital filters, FFTs, matrix operations, and DMAs. With its 16 bit architecture, registers, dual ports, and speed, the 48-pin ADSP-1410 improves performance and reduces board space substantially relative to bit-slice solutions.

The ADSP-1410's architecture features a 16-bit ALU, a comparator, and 30 16-bit registers. The registers are organized into four files: sixteen address (R) registers, six offset (B) registers, four compare (C) registers, and four initialization (I) registers.

The ADSP-1410 rapidly executes key address generating operations. In a single instruction cycle, the device can

- output a 16-bit memory address,
- modify this memory address, and,
- detect when the address value has moved to or beyond a pre-set boundary and conditionally loop back to the top of a circular buffer.

Consequently, circular buffers and modulo addressing for data memories can be implemented without overhead.

The ADSP-1410's 10-bit microcode instructions include commands for looping, register read/writes, internal data transfers, logical/shift operations. Instructions are normally supplied from an external source. However, an internal Alternate Instruction Register (AIR) can provide the instruction under external control, allowing microcode to be conserved in many applications.

The ADSP-1410 has a 16-bit address (Y) port for outputting addresses and a 16-bit data (D) port for I/O between internal and external registers. Also, an internal path allows external data, provided via the D port, to serve as an ALU source and/or to be directly output over the Y port for a DMA capability.

Double-precision (30 bit), single-cycle addressing can be performed by cascading two ADSP-1410's, with the MSB of each chip's D and Y port dedicated to interchip communication. Alternatively, a single AG can provide double-precision addresses at a rate of one per two clock cycles.

The Look-Ahead™ pipeline eliminates the need for an external microcode pipeline register by internaly latching instructions and addresses, microcode bits may be directly routed to the ADSP-1410 from microcode memory. Logically the Look Ahead™ pipeline is split into two havles: the first, located at the instruction (and data) port, and the second, located at the address port. Each half of the pipeline (input vs. output) has a transparent latch which operates out of phase with the other: the address latch is transparent during the first half of the cycle (clock HI), while the input latches (instruction and data) are transparent during the second half of the cycle (clock LO). This complementary arrangement allows new instructions to be decoded (in preparation for the following cycle) while the program address for the current cycle is held steady.

ADSP-1410 OVERVIEW

Digital Signal Processing (DSP) and array processing systems require fast, flexible address generation circuitry. An Address Generator (AG) supplies the address of a location in data or coefficient memory. The value residing at the specified address is fetched and fed to an arithmetic unit for processing. The AG must then modify the address pointer in anticipation of the next data fetch. For algorithms that repetitively loop through data buffers, the AG may need to compare the address to a buffer end and conditionally loop back to the top of the buffer. Finally, to maximize throughput, an AG must perform its addressing tasks rapidly and without overhead.

With the ADSP-1410, 16 bit pointers to memory are stored in an address (R) register file. Since an AG must track several pointers concurrently, sixteen R registers, denoted R_n, are provided. If we denote Y as the address port, the operation "$Y \leftarrow R_n$" corresponds to the AG supplying an address from register R_n.

After supplying an address, the AG must update the pointer for the next memory fetch. The updating may be as simple as an increment but, more generally, involves adding or subtracting an arbitrary offset value. Also, algorithms generally access several different offset values. To this end, the AG provides six offset registers, denoted B_m, and can execute in a single-cycle the core operation

$$Y \leftarrow R_n,\ R_n \leftarrow R_n + B_m$$

In DSP applications, data arrays are often addressed as circular buffers. That is, when addressing reaches the buffer end, it wraps back to the beginning of the buffer. To implement this looping, the AG compares the supplied address to one of four compare registers, denoted C_j. If the address has moved to or beyond the end of the boundary ($R_n \geq C_j$), the device can transfer an initialization register value, denoted I_j, to the register ($R_n \leftarrow I_j$), otherwise, it is updated in normal fashion ($R_n \leftarrow R_n + B_m$). To minimize overhead, the AG can execute normal updates while also performing conditional re-initializations, again, in one core operation

$$Y \leftarrow R_n,\ \text{IF } (R_n \geq C_j)\ R_n \leftarrow I_j,\ \text{ELSE } R_n \leftarrow R_n + B_m$$

Since the above instruction handles the looping required of circular buffer addressing, it is termed a looping instruction. To a large extent, the ADSP-1410's architecture and instruction set revolve around efficient implementation of this instruction. However, many variations of this instruction are supported on the device and spelled out in the following sections.

ADDRESS SOURCES
- Sixteen internal R registers
- External data provided over the D port

OFFSET SOURCES
- Six internal B registers
- Data Port

OFFSET OPERATIONS
- Increment ($R_n \leftarrow R_n + 1$)
- Decrement ($R_n \leftarrow R_n - 1$)
- Add Offset ($R_n \leftarrow R_n + B_m$)
- Subtract Offset ($R_n \leftarrow R_n - B_m$)
- Single-Bit Left/Right Shifts
- Logical Operations (AND, OR, XOR)

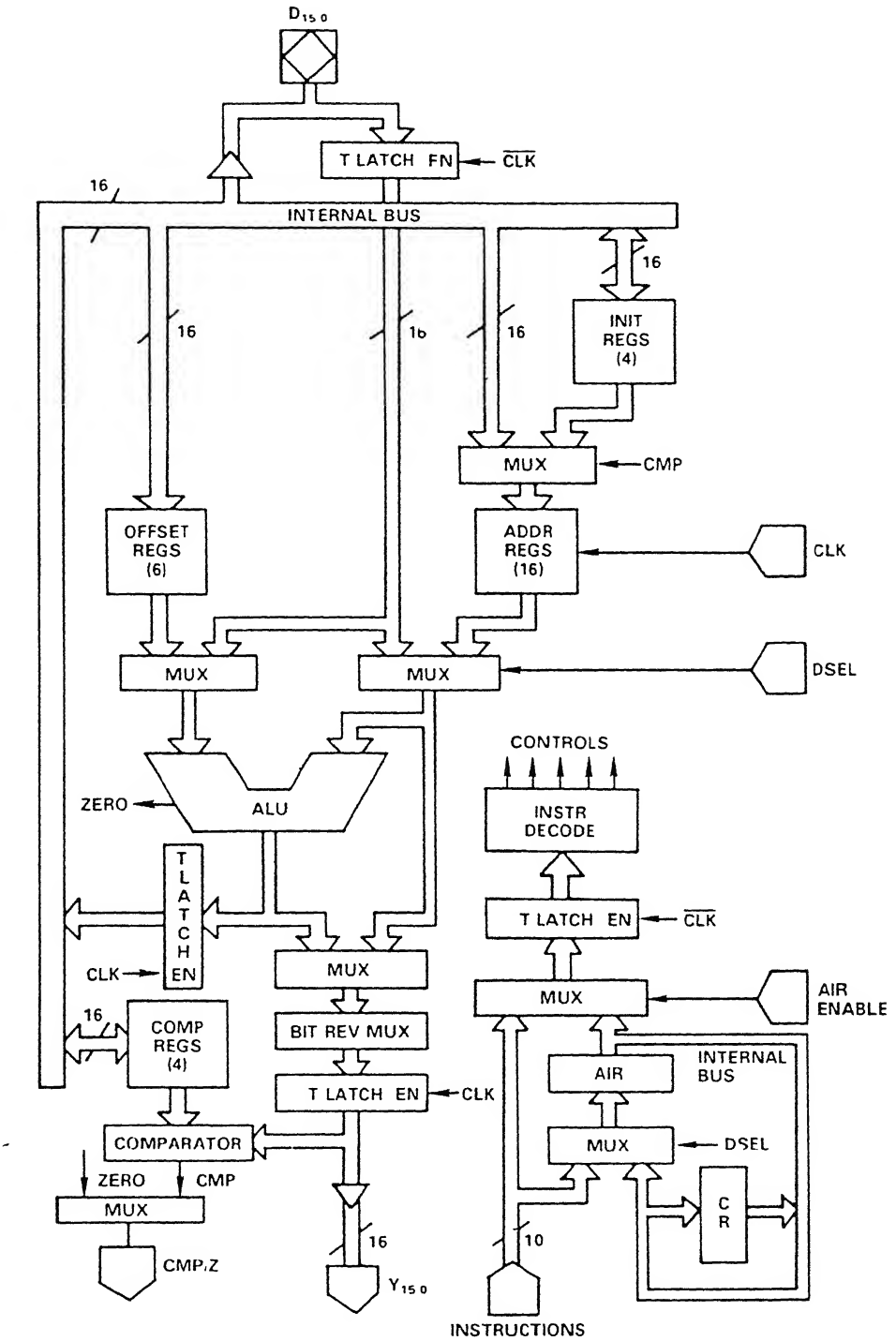
CONDITIONAL RE-INITIALIZATION
- Independent Inhibit/Enable for each of four initialization registers
- Conditional AIR execution (used for true modulo addressing)

OUTPUT/UPDATE SEQUENCE
- Normal (Pre-Update) Mode (output the address before update)
- Post-Update Mode (output the address after update)

PRECISION
- Single chip supplies 16-bit addresses
- Two chips cascaded provide 30-bit addresses
- One chip provides 30-bit addresses in two cycles

ADSP-1410 PIN ASSIGNMENTS

PIN NAME	DESCRIPTION
$Y_{15} - Y_0$	The address (Y) output port. In single-chip/double-precision mode, the MSB (Y_{15}) indicates whether the supplied address is the MSW or LSW (see Precision Modes). In two-chip/double-precision mode, the MSB conveys the carry/shift bit from the Least Significient (LS) to the Most Significient (MS) chip.
$D_{15} - D_0$	The bi-directional data (D) port. In two-chip/double-precision addressing mode, the MSB (D_{15}) of this port conveys CMP status from the partner chip.
$I_9 - I_0$	The instruction port.
CMP/Z	A dual function pin. Looping instructions, which compare address register values to compare register values, assert this pin HI to convey CMP status if i) $R \geq C$ for positive offsets, or ii) $R \leq C$ for negative offsets. Logical/Shift instructions assert this pin HI to convey the ZERO status of the result.
DSEL	Data Select control. Asserting this control HI causes data set up on the data port to substitute for the R value specified in the instruction.
AIR Enable	Alternate Instruction Register control. Asserting this control HI causes the device to execute an instruction stored in the internal AIR, rather than the instruction set up on the instruction port.
CLK	Clock.
V_{dd}	+5 Volt Power Supply.
GND	Ground.



ADSP 1410 Functional Block Diagram

ANALOG DEVICES

High-Speed 64-Bit IEEE Floating Point Multiplier and ALU

PRELIMINARY TECHNICAL DATA

FEATURES

- **Implements a Full Floating Point Processor Solution, Handling 32-Bit and 64-Bit Floating Point Numbers**
- **Fully Compatible with IEEE Standard 754**
- **Three Data Formats**
  - **32-Bit Single Precision Floating Point**
  - **64-Bit Double Precision Floating Point**
  - **32-Bit Fixed Point**
- **Fast**
  - **Single Precision Throughput of 100ns/ 10 MEGAFLOPS for All Operations**
  - **Double Precision Throughput of 100ns/10 MEGAFLOPS (ADSP-3220) 400ns/2 5 MEGAFLOPS (ADSP 3210)**
  - **32-Bit Fixed Point Throughput of 100ns/10MHz for All Operations**
- **Flexible I/O Structures Support Full Data Transfer Rate**
  - **ADSP-3220 Supports Three- and Two-Port Structures**
  - **ADSP-3210 Supports Two-Port Structures**
- **One Internal Pipeline Stage in Each Part**
- **Multiple Input Registers Associated with Each Input Port**
- **400mW Max Power Dissipation Per Chip with CMOS Technology**
- **Fully Registered Inputs, Outputs, and Control Signals**
- **Three-State Outputs with Separate Enables**
- **100-Lead Pin Grid Array (ADSP-3210)**
- **144-Lead Pin Grid Array Package (ADSP-3220)**

NERAL DESCRIPTION

e ADSP-3210 Floating Point Multiplier and ADSP-3220 ating Point ALU are high-speed, low-power arithmetic pro- ors with data format conforming to IEEE Standard 754

ADSP-3210 and ADSP-3220 comprise the basic elements to lement a high-speed numerical processor with operations on e data formats 32-bit IEEE single-precision floating point, it IEEE double-precision floating point, and 32-bit two's plement fixed point

icated in CMOS, the ADSP-3210 and ADSP-3220 provide h-speed (100ns cycle time at 70°C) and low power (less 400mW power dissipation per chip) floating point processor ion The processors offer very high throughput for all three formats 32 bit IEEE Single-Precision results produced 100ns, 64-bit IEEE Double-Precision results every 100ns P-3220) and every 400ns (ADSP-3210), and fixed point s every 100ns

The chips' data formats and floating point operations conform to the proposed IEEE Standard 754, Draft 10 0, assuring complete software portability for computational algorithms adhering to the Standard The chips support all four rounding modes in the Standard for all three data formats All four exception conditions detected—overflow, underflow, invalid operation, and inexact result—are provided as dedicated status pins, minimizing response time of the external system to exceptions

The ADSP-3210 and ADSP-3220 have a powerful instruction set designed for a systems-level implementation of function calculations Specific instructions are included to facilitate such functions as table lookup, floating point divide and square root, quadrant normalization for trig functions, and operations on denormals

The ADSP-3210 Block Diagram shows the Floating Point Multiplier's two-port structure one 32-bit input port and one 32-bit output port Two 32-bit registers are available for each of the A and B operands Data inputs and outputs transfer at twice the cycle rate, performing two 32-bit input operations and supplying an entire 64-bit or 32-bit output product on every cycle All inputs and outputs are registered

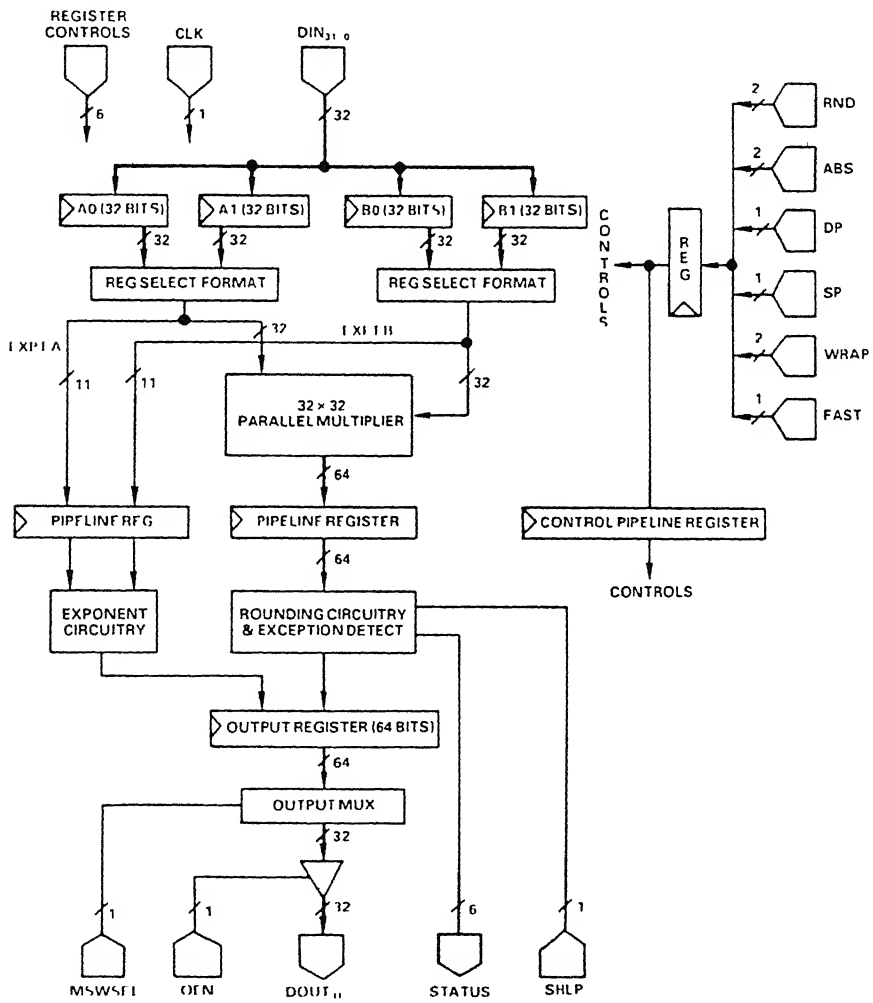
The ADSP-3220 Block Diagram shows the three-port structure of the Floating Point ALU two 32-bit input ports and one 32 bit output port The ADSP-3220 can be configured to have one or two input ports Four input registers are available for each of the A and B operands, each input port can load any of the eight input registers

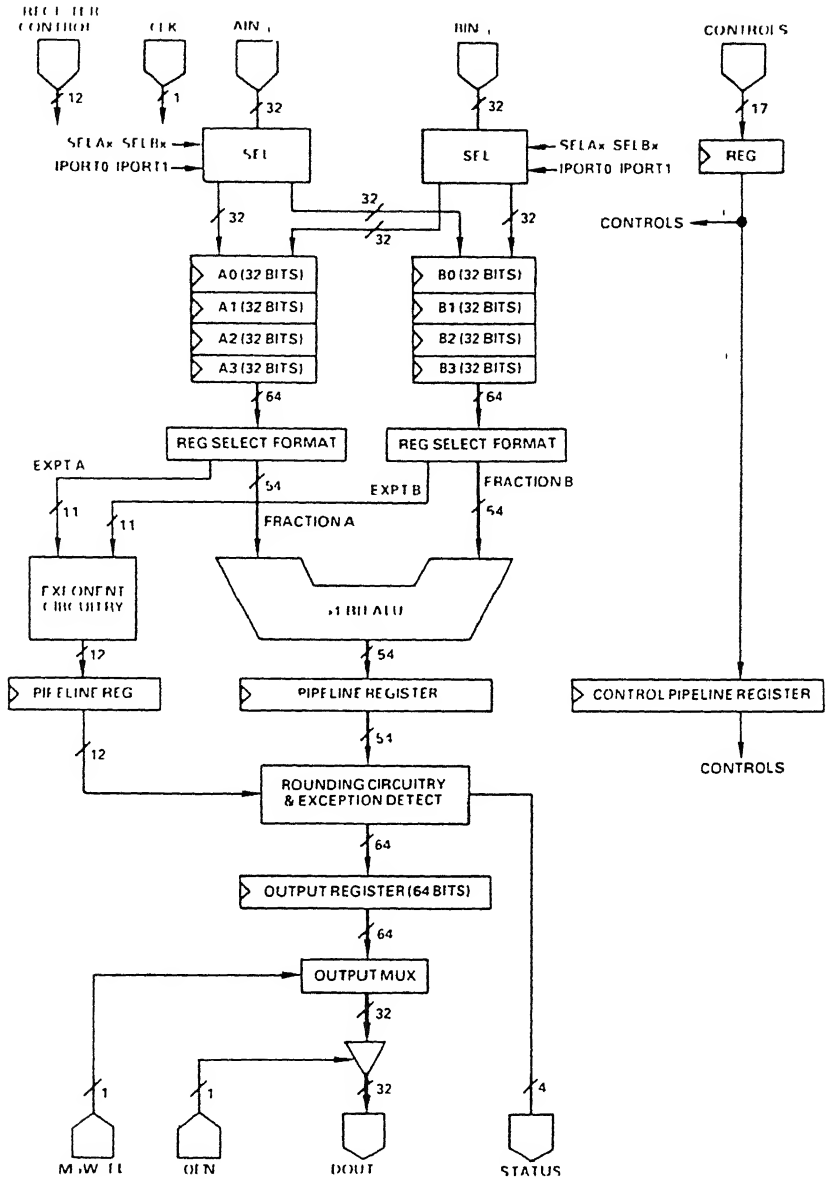
The ADSP-3220's three port structure accommodates the full 10 MEGAFLOPS throughput rate for Double Precision (DP) operations, loading two 64 bit operands on each cycle and operating internally with 64 bit wide data paths. The ADSP-3210's two port structure accommodates the full DP throughput rate (4 cycles/2.5 MEGAFLOPS), performing all four DP cross-products in 4 cycles with its 32 × 32 multiplier array.

Fixed point addition or subtraction accepts two 32 bit two's complement operands and produces a 32 bit two's complement result every 100ns. Fixed point multiplication produces a 64 bit two's complement product every 100ns. The 64 bit product on the ADSP-3210 may be shifted left by one bit on output to eliminate a redundant sign bit in the most significant word of the product.

The ADSP-3210 and ADSP-3220 support the gradual underflow provisions of the IEEE standard. A FAST mode is included on each chip, which sets results less than the IEEE normalized format to zero. FAST mode simplifies underflow exception handling while retaining all the other benefits of the high dynamic range and precision in the IEEE Floating Point format.

ADSP-3210 FLOATING POINT MULTIPLIER




REGISTER CONTROL
CLK
AIN
BIN
CONTROLS
12
1
32
32
17
SELAx SELBx
IPORT0 IPORT1
SEL
SEL
SELAx SELBx
IPORT0 IPORT1
REG
32
32
32
32
CONTROLS
A0 (32 BITS)
A1 (32 BITS)
A2 (32 BITS)
A3 (32 BITS)
B0 (32 BITS)
B1 (32 BITS)
B2 (32 BITS)
B3 (32 BITS)
64
64
REG SELECT FORMAT
REG SELECT FORMAT
EXPT A
EXPT B
FRACTION B
54
11
11
54
FRACTION A
EXPONENT CIRCUITRY
54
12
PIPELINE REG
PIPELINE REGISTER
CONTROL PIPELINE REGISTER
54
12
CONTROLS
ROUNDING CIRCUITRY & EXCEPTION DETECT
64
OUTPUT REGISTER (64 BITS)
64
OUTPUT MUX
32
1
1
32
4
MS,W TE
OEN
DOUT
STATUS

PIN DEFINITIONS

(Positive Logic Naming Convention is Used Throughout)

Unless otherwise noted, pin definitions apply to both the ADSP-3210 and ADSP-3220

SP denotes 32-bit Single-Precision floating point
DP denotes 64 bit Double Precision floating point
FIXED denotes 32 bit fixed point format
x denotes the Input Register Number

Data Lines

DIN_{31} – DIN_0 (ADSP-3210 only) 32 Data Input Pins DIN_{31} is the most significant bit (MSB)

AIN_{31} – AIN_0 (ADSP-3220 only) 32 Data Input pins AIN_{31} is the MSB

BIN_{31} – BIN_0 (ADSP-3220 only) 32 Data Input pins BIN_{31} is the MSB

$DOUT_{31}$ – $DOUT_0$ 32 Data Output pins $DOUT_{31}$ is the MSB

Asynchronous Input Control Lines

IPORT1, IPORT0 (3220 only) Controls which select the port configuration, source input port(s), and destination input registers These controls should be hardwired to the desired port configuration

IPORT1	IPORT0	Port Configuration	Source and Destination	
0	0	Two Port	AIN_{31-0}	B Registers
			BIN_{31-0}	A Registers
1	0	One Port	AIN_{31-0}	source for all registers
0	1	One Port	BIN_{31-0}	source for all registers
1	1	Two Port	AIN_{31-0}	A Registers
			BIN_{31-0}	B Registers

Registered Input Control Lines

All registered input controls are latched on the rising edge of the clock All controls are clocked with the intermediate results at the internal pipeline register, keeping the control lines associated with the proper operands

CLK Clock Input The rising edge of CLK latches the controls for an operation, initiates the operation, and clocks input data into the selected register(s) The falling edge of CLK only clocks input data into the selected register(s)

SFI Ax, SFI Bx Input Register Select controls for loading the input registers with the input port data

ADSP-3210	Signal	Action
	SFI A0	Load A0 on ⎍
	SFI A1	Load A1 on ⎍
	SELB0	Load B0 on ⎍
	SELB1	Load B1 on ⎍

ADSP-3220 Signal	Action for Two-Input Port Configuration[1]	Action for One-Input Port Configuration[1]
SELA0	Load A0 on ⎍	Load A0 on ⎍
SELA1	Load A1 on ⎍	Load A1 on ⎍
SFI A2	Load A2 on ⎍	Load A2 on ⎍
SFI A3	Load A3 on ⎍	Load A3 on ⎍
SFI B0	Load B0 on ⎍	Load B0 on ⎍
SELB1	Load B1 on ⎍	Load B1 on ⎍
SFI B2	Load B2 on ⎍	Load B2 on ⎍
SFI B3	Load B3 on ⎍	Load B3 on ⎍

NOTE
[1](ADSP 3220 only) IPORT1, IPORT0 control one input or two-input port configurations

RDA1, RDA0, RDB1, RDB0 Input Operand Select controls selecting the operands for the operation

ADSP-3210	State	Register Selected
RDA0	0	A1
	1	A0
RDB0	0	B1
	1	B0

(ADSP 3210 only) For double precision operations, both RDA0 and RDB0 must be HIGH at the start of the operation After initiation of the DP operation, the ADSP-3210 controls the DP multiplication, and RDA0 and RDB0 are ignored until completion

ADSP-3220	State	Register Selected
RDA1 RDA0	0 0	A2
	0 1	A3
	1 0	A0
	1 1	A1
RDB1 RDB0	0 0	B2
	0 1	B3
	1 0	B0
	1 1	B1

(ADSP-3220 only) For double precision operations, only RDA1 and RDB1 select the DP operand pairs, and RDA0 and RDB0 must be HIGH

ABSA, ABSB Absolute Value controls ABSA causes the chip to convert the selected A operand to its absolute value before performing the operation, ABSB causes the B operand to take on its absolute value On the ADSP-3220 Absolute Value is available for all SP, DP, and FIXED operands On the ADSP-3210, Absolute Value is only available for SP and DP operands

RND1, RND0 Rounding mode controls See Method of Operation for explanation of rounding The round control modes are

RND0	RND1	Rounding Mode
0	0	Round to Nearest Number
0	1	Round to Plus Infinity
1	0	Round to Zero
1	1	Round to Minus Infinity

SP (ADSP-3210 only) Single Precision mode (active HIGH) Selects 32-bit Single Precision format for both operands and product

DP (ADSP-3210 only) Double Precision mode (active HIGH) Selects 64-bit Double Precision format for both operands and product

(ADSP-3210 only) If neither SP nor DP is HIGH on the rising edge of CLK, the 3210 will operate in FIXED mode, multiplying two 32-bit Fixed Point two's complement operands and producing a 64-bit two's complement product Asserting both SP and DP is an illegal state, causing an indeterminate output

$I_8 - I_0$ (ADSP-3220 only) Instruction control lines For ADSP 3220 instruction set, see Method of Operation section Selection of SP, DP, or FIXED data formats is explicit in the ADSP-3220 Instruction word

FAST Fast mode pin When FAST is active (HIGH), an underflow will return a result of all zeroes for SP and DP operations FAST has no effect on FIXED mode operations If FAST is inactive (LOW), then the chips produce IEEE-compatible outputs for denormalized and underflowed results

(ADSP-3210 only) A floating point underflow result with FAST LOW (IEEE mode) will return a "wrapped number" correct fraction and sign, with exponent a two's complement negative number[1] With FAST HIGH, denormalized inputs are forced to zero for the operation

(ADSP-3220 only) FAST LOW generates proper denormalized outputs for underflowed results With FAST HIGH, denormalized and underflowed results are forced to zero, but denormal inputs are not modified before performing the operation FAST HIGH also forces to zero the denormal and underflowed results of DP to SP conversions and comparison operations

WRAPA, WRAPB (ADSP-3210 only) Control pin that labels the selected A or B register as a denormalized ("wrapped") number The multiplier then treats the input's exponent as a two's complement negative number[1]

SHLP (ADSP-3210 only) Control pin to shift left a 64-bit Fixed Point product When HIGH, SHLP shifts the 64-bit output register left by one bit on output (eliminating a redundant sign bit in the two's complement product), and shifts a zero into the LSB SHLP has no shift effect on floating point outputs SHLP is clocked with the operands at each level of the pipeline, therefore a change in the state of SHLP will take effect on the output register one cycle later

Non-Registered (Unclocked) Control Lines

MSWSEL When true (HIGH), MSWSEL selects the most significant 32 bits of the output register on $DOUT_{31-0}$ When MSWSEL is LOW, the least significant 32 bits are selected for output MSWSEL takes effect on the output register immediately (it is not latched) For SP and Logical operations, the proper output is selected with MSWSEL HIGH

OEN Output data enable OEN HIGH enables data on $DOUT_{31-0}$, OEN LOW causes the data output pins to be in a high-impedance state

$\overline{\text{RESET}}$ Reset control pin When $\overline{\text{RESET}}$ goes active (LOW), the chip is reset internally $\overline{\text{RESET}}$ clears all control functions in the chip, sets all status flags to zero, but does not clear the input registers $\overline{\text{RESET}}$ should be activated on power up, ensuring proper initialization of the chip

Status Outputs

All status outputs are active HIGH All status outputs except DENORM are paired with their operands through the pipeline, so they become true when the corresponding result is clocked into the output register For specific conditions causing each exception, see Exceptions and Status Outputs section

INEXO Inexact Result status output, generated when the result could not be expressed exactly in the destination format without loss of accuracy

OVRFLO Overflow status output, generated when the result of a SP or DP operation is greater than the maximum representable number in the the destination format before rounding

UNDFLO Underflow status output, generated when the result of a SP or DP operation is less than the minimum representable number in the destination format When underflow occurs, the result produced at the output depends on the state of FAST (see FAST Pin Description)

INVALOP Invalid Operation status output, generated when an invalid operation as specified in the IEEE Standard 754 occurred (e g , $0 \times \infty$)

DENORM (ADSP-3210 only) Denormal status output This signal is active (HIGH) when a denormalized number is detected as an input operand, and informs the system that the input must be wrapped by the ADSP 3220 if the multiplier is to handle it properly[1] DENORM becomes valid on the clock cycle after the operand(s) are loaded into the multiplier array The multiplier will set the denormal to zero and complete the multiplication A denormal input causes the same status flags to appear as a zero input would

RNDCARO (ADSP-3210 only) Round Propagated status output This signal can only occur when the UNDFLO status output on the ADSP-3210 also becomes true RNDCARO indicates that a carry bit propagated across the fraction's rounding boundary when the fraction was rounded to the destination format RNDCARO is used in conjunction with INEXO to enable the ADSP-3220 to unwrap a wrapped number correctly[1]

Registered Status Inputs (ADSP-3220 only)

RNDCARI Round Propagated status input Same as RNDCARO on the ADSP-3210, except that on the ADSP 3220 this is an input The controlling system provides the RNDCARI input to the ADSP-3220 when performing the UNWRAP function[1]

INLXIN Inexact Input Same as INLXO on the ADSP-3210, except that on the ADSP-3220 this is an input The controlling system provides the INEXIN input to the ADSP-3220 when performing the UNWRAP function[1]

Other Signals

VDD +5 volt power supply

GND Ground

[1]See Application Note on Handling IEEE Exceptions

[1]See Application Note on Handling IEEE Exceptions

Integrated Device Technology, Inc.

CMOS PARALLEL FIRST-IN/FIRST-OUT FIFO 1024 x 9-BIT

IDT7202SA/LA

FEATURES

- First-In/First Out dual port memory
- 1024 x 9 organization
- Low power consumption
- Ultra high speed—35ns cycle time (28.5MHz)
- Asynchronous and simultaneous read and write
- Fully expandable by both word depth and/or bit width
- Pin compatible with Mostek MK4501 but with Half Full Flag capability
- Allows for deep word structure (1024) without expansion
- Half-Full Flag capability in single device mode
- Master/Slave multiprocessing applications
- Bidirectional and rate buffer applications
- Empty and Full warning flags
- Auto retransmit capability
- High performance CEMOS™ technology
- Available in Plastic DIP, CERDIP, 300 mil sidebraze THINDIP, LCC, PLCC and Flatpack
- Military product compliant to MIL-STD-883, Class B

DESCRIPTION

The IDT7202A is a dual-port memory that utilizes a special First In/First Out algorithm that loads and empties data on a first-in/first-out basis. The device uses Full and Empty flags to prevent data overflow and underflow and expansion logic to allow for unlimited expansion capability in both word size and depth.

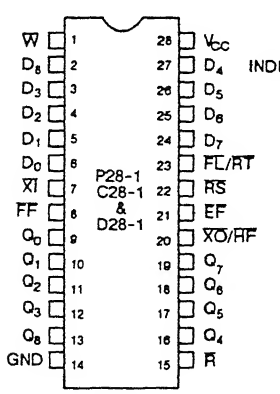
The reads and writes are internally sequential through the use of ring pointers, with no address information required to load and unload data. Data is toggled in and out of the device through the use of the Write ($\overline{W}$) and Read ($\overline{R}$) pins. The device has a read/write cycle time of 35ns (28.5MHz).

The device utilizes a 9-bit wide data array to allow for control and parity bits at the user's option. This feature is especially useful in data communications applications where it is necessary to use a parity bit for transmission/reception error checking. It also features a Retransmit ($\overline{RT}$) capability that allows for reset of the read pointer to its initial position when $\overline{RT}$ is pulsed low to allow for retransmission from the beginning of data. A Half-Full Flag is available in the single device mode and width expansion modes.

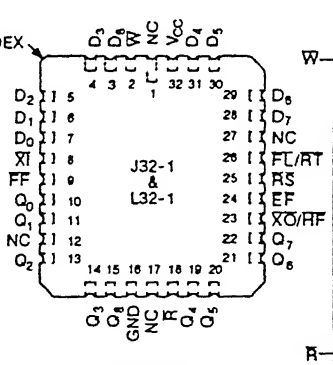
The IDT7202A is fabricated using IDT's high speed CEMOS technology. It is designed for those applications requiring asynchronous and simultaneous read/writes in multiprocessing and rate buffer applications. The 1024 x 9 organization of the IDT7202A allows a 1024 deep word structure without the need for expansion.

Military grade product is manufactured in compliance with the latest revision of MIL-STD-883, Class B.

PIN CONFIGURATIONS



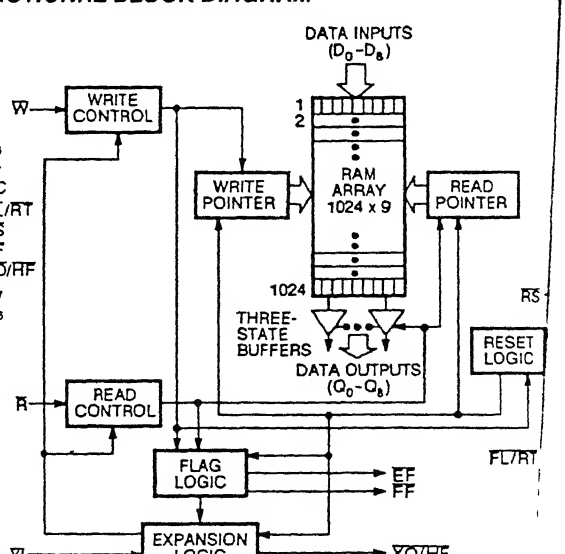
DIP
TOP VIEW



LCC/PLCC
TOP VIEW

CONSULT FACTORY FOR FLATPACK PINOUT

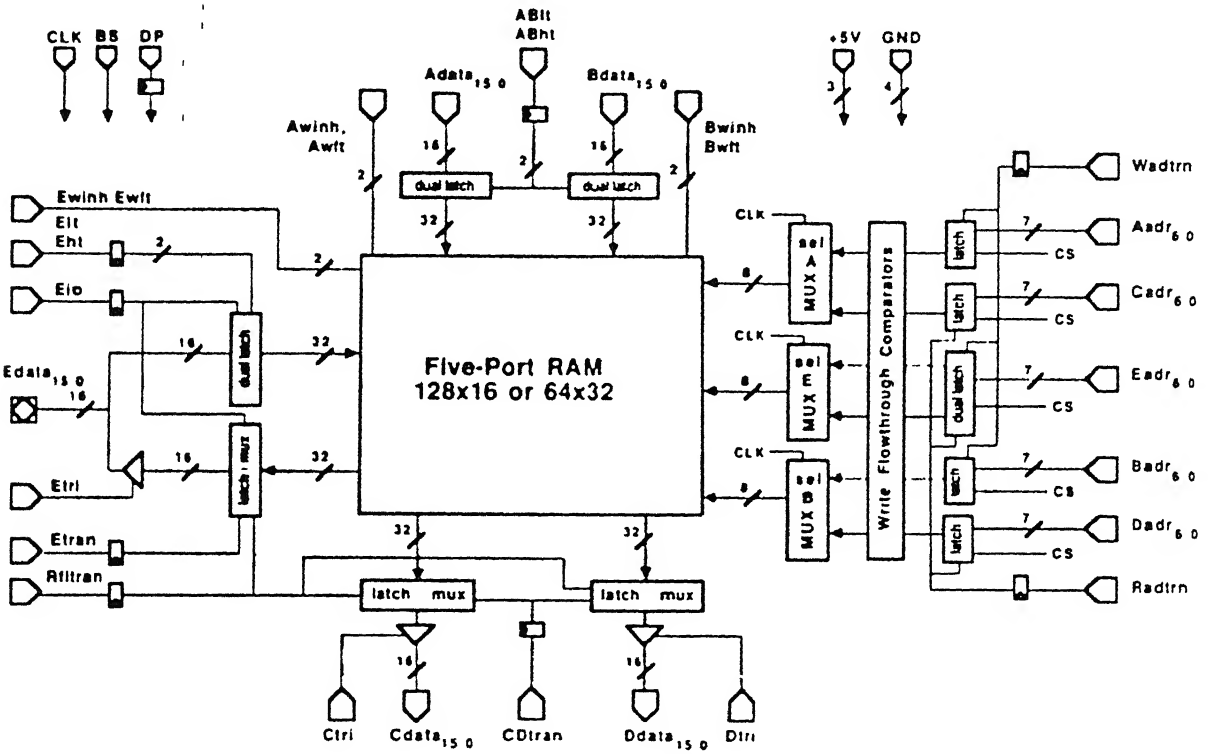
FUNCTIONAL BLOCK DIAGRAM



FUNCTIONAL DESCRIPTION

The ADSP 3128 Multiport Register File consists of a high-speed static RAM (configurable as either 128 × 16 or 64 × 32) surrounded by the latches and control logic needed for simple system interfacing (see Figure 1). Six internal data paths, all 32 bits wide, connect this RAM with multiplexers (muxes) and latches. Three are read data paths, three are write data paths. Three 7 bit internal address paths connect this RAM with muxes and address latches. These three address paths are time-multiplexed to allow the presentation of six addresses to the RAM per cycle. Hence, up to a total of six reads from and writes to the RAM are possible per cycle. Because of the abundance of data paths, many of which can transfer data twice per cycle, many combinations of six reads and writes are possible.

Three addresses are presented to RAM in clock HI from the Aadr, Badr, and Eadr address latches. Normally in clock HI, these are RAM write addresses. They are prioritized in case of conflict. Three addresses are presented to RAM in clock LO from the Cadr, Dadr, and Eadr address latches. Normally in clock LO, these are RAM read addresses. Three simultaneous reads from the same RAM location are possible for normal, clock LO reads. The EadrPort feeds both a write (clock HI) address latch and a read (clock LO) address latch, which can be independently set to latched or transparent modes.



ADSP 3128 Multiport Register File Functional Block Diagram

BS	DP	AB & Elt	AB & Eht	A & B & Einh	A & B & Ewft	Description
0	X	X	X	X	X	Disable chip (consistent with pipelines) but advance pipelines with clock cycle
1	0	0	0	X	X	Register write data at A & B or Edata input latches on falling edge
1	0	0	1	X	X	Hold most recent data at A & B or Edata input latches for the next cycle
1	0	1	0	X	X	Latch write data at A & B or Edata input latches at clock HI
1	0	1	1	X	X	Make transparent A & B or Edata input latches
1	X	X	X	0	X	Allow write to RAM from the A, B, and Edata input latches
1	X	X	X	1	X	Inhibit write to RAM from the A, B, and Edata input latches
1	0	X	X	0	0	Normal write to RAM from the A or B or Edata input latches during clock HI
1	0	X	X	X	1	Flow-through write (transparent) to RAM from the A or B or Edata input latches during clock LO when write/read addresses are equal
1	1	0	0	X	X	Early Input to A & B or Edata input latches register LSW on falling edge to input latches and latch MSW to input latches in clock HI
1	1	0	1	X	0	Late Input to A & B or Edata input latches latch LSW to input latches in clock HI and make input latches transparent for MSW in clock HI
1	1	0	0	X	1	Undefined
1	1	1	X	X	X	Hold most recent data at A & B or Edata input latches for the next cycle
1	1	1	1→0	X	X	Edata Slow Input register LSW to Edata input latch on next falling edge (Eht only)
1	1	1	0→1	X	X	Edata Slow Input register MSW to Edata input latch on next falling edge (Eht only)

Table I ADSP-3128 Summary of Data Input and Write Control Modes

BS	DP	CD & Etran	Rfltran	C & D & Eut	Eio	Description
0	X	X	X	X	X	Disable chip (consistent with pipelines) but advance pipelines with clock cycle
1	X	X	X	0	X	Drive data from output latches through C or D or Edata-Port
1	X	X	X	1	X	Three-state (high impedance) output C or D or Edata-Port
1	0	0	X	X	X	Register data from RAM to C& D or Edata output latches on rising edge
1	0	1	0	X	X	C & D or Edata output latches are transparent clock LO latched clock HI
1	0	1	1	X	0	C & D or Edata RAM output latches are fully transparent for both phases If Awinh/Bwinh/Ewinh = 1, an additional read(s) can be performed at C/D/Edata, respectively
1	0	X	X	X	0	Edata-Port is configured for one read, one write, or two reads per cycle
1	0	X	X	X	1	Edata Port is configured for both a read and a write every cycle, read Eadr is registered on falling edge and write Eadr is registered on rising edge
1	1	0	0	X	0	Configured for Late Read at C& D or Edata-Port register LSW & MSW from RAM to output latches on rising edge, output LSW in clock HI, output MSW on next clock LO
1	1	1	0	X	0	Configured for Early Read at C&D or Edata-Port output LSW from RAM through transparent output latches in clock LO, latch MSW to output latches and output in clock HI
1	1	0	0	X	1	Configured for Edata Slow Read hold RAM read data at Edata output latch, output LSW at clock HI
1	1	1	0	X	1	Configured for Edata Slow Read hold RAM read data at Edata output latch, output MSW at clock HI
1	1	X	1	X	X	Undefined

Table II ADSP-3128 Summary of Data Read and Output Control Modes

BS	DP	Wadtrn	Radtrn	A/B/C/D/Eadr$_6$ (Port Select)	Description
0	X	X	X	X	Disable chip (consistent with pipelines) but advance pipelines with clock cycle
1	X	0	X	X	Latch A or B or Eadr write addresses at clock HI*
1	X	1	X	X	A or B or Eadr write address latches are transparent*
1	X	X	0	X	Register C or D or Eadr read address latches on the rising edge*
1	X	X	1	X	C or D or Eadr read address latches are transparent*
X	1	X	X	0	Disable A/B C/D/Edata-Port
1	1	X	X	1	Enable A B C D/Edata Port

*Note Eio = 1 overrides Wadtrn and Radtrn at the Eadr Port (only) That is, when Eio = 1, Eadr write addresses are registered on rising edges and read addresses are registered on falling edges regardless of the state of Wadtrn and Radtrn The other four address ports are unaffected by Eio and always behave as described in this table

ADSP 3128 Summary of Address Control Modes

PRELIMINARY TECHNICAL